

Satisfiability Modulo Theories

Roberto Sebastiani

DISI, University of Trento, Italy.
roberto.sebastiani@disi.unitn.it
<http://www.disi.unitn.it/~rseba>

j. w. w.

Gilles Audemard, Clark Barrett, Marco Bozzano, Roberto Bruttomesso, **Alessandro Cimatti**,
Anders Franzén, **Alberto Griggio**, Tommi Junttila, Ziyad Hanna, Zurab Khasidashvili,
Arthur Kornilowicz, Alexander Nadel, Silvio Ranise, Peter vanRossum, Sanjit Seshia,
Bas Schaafsma, Stephan Schultz, Cristian Stenico, Cesare Tinelli, Silvia Tomasi,...

22nd International Joint Conference on Artificial Intelligence – IJCAI 11

Barcelona, Spain, July 16th 2011.

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Satisfiability Modulo Theories (SMT($\mathcal{T}$))

Satisfiability Modulo Theories (SMT($\mathcal{T}$))

The problem of deciding the satisfiability of (typically quantifier-free) formulas in some decidable first-order theory $\mathcal{T}$

- $\mathcal{T}$ can also be a *combination of theories* $\bigcup_i \mathcal{T}_i$.

SMT($\mathcal{T}$): theories of interest

Some theories of interest (e.g., for formal verification)

- *Equality and Uninterpreted Functions ($\mathcal{EUF}$):*
 $((x = y) \wedge (y = f(z))) \rightarrow (g(x) = g(f(z)))$
- *Difference logic ($\mathcal{DL}$):* $((x = y) \wedge (y - z \leq 4)) \rightarrow (x - z \leq 6)$
- *UTVPI ($\mathcal{UTVPI}$):* $((x = y) \wedge (y - z \leq 4)) \rightarrow (x + z \leq 6)$
- *Linear arithmetic over the rationals ($\mathcal{LA}(\mathbb{Q})$):*
 $(T_\delta \rightarrow (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0)) \wedge (\neg T_\delta \rightarrow (s_1 = s_0))$
- *Linear arithmetic over the integers ($\mathcal{LA}(\mathbb{Z})$):*
 $(x := x_l + 2^{16}x_h) \wedge (x \geq 0) \wedge (x \leq 2^{16} - 1)$
- *Arrays ($\mathcal{AR}$):* $(i = j) \vee read(write(a, i, e), j) = read(a, j)$
- *Bit vectors ($\mathcal{BV}$):*
 $x_{[16]}[15 : 0] = (y_{[16]}[15 : 8] :: z_{[16]}[7 : 0]) \ll w_{[8]}[3 : 0]$
- *Non-Linear arithmetic over the reals ($\mathcal{NLA}(\mathbb{Q})$):*
 $((c = a \cdot b) \wedge (a_1 = a - 1) \wedge (b_1 = b + 1)) \rightarrow (c = a_1 \cdot b_1 + 1)$

Satisfiability Modulo Theories (SMT($\mathcal{T}$)): Example

Example: $\text{SMT}(\mathcal{LA}(\mathbb{Z}) \cup \mathcal{EUF} \cup \mathcal{AR})$

$$\varphi \stackrel{\text{def}}{=} (d \geq 0) \wedge (d < 1) \wedge \\ ((f(d) = f(0)) \rightarrow (\text{read}(\text{write}(V, i, x), i + d) = x + 1))$$

- involves arithmetical, arrays, and uninterpreted function/predicate symbols, plus Boolean operators
 - Is it consistent?
 - No:

$$\begin{array}{ll} \varphi & \\ \implies_{\mathcal{LA}(\mathbb{Z})} & (d = 0) \\ \implies_{\mathcal{EUF}} & (f(d) = f(0)) \\ \implies_{\text{Bool}} & (\text{read}(\text{write}(V, i, x), i + d) = x + 1) \\ \implies_{\mathcal{LA}(\mathbb{Z})} & (\text{read}(\text{write}(V, i, x), i) = x + 1) \\ \implies_{\mathcal{LA}(\mathbb{Z})} & \neg(\text{read}(\text{write}(V, i, x), i) = x) \\ \implies_{\mathcal{AR}} & \perp \end{array}$$

Satisfiability Modulo Theories (SMT($\mathcal{T}$)): Example

Example: $\text{SMT}(\mathcal{LA}(\mathbb{Z}) \cup \mathcal{EUF} \cup \mathcal{AR})$

$$\varphi \stackrel{\text{def}}{=} (d \geq 0) \wedge (d < 1) \wedge \\ ((f(d) = f(0)) \rightarrow (\text{read}(\text{write}(V, i, x), i + d) = x + 1))$$

- involves arithmetical, arrays, and uninterpreted function/predicate symbols, plus Boolean operators
 - Is it consistent?
 - No:

$$\begin{array}{ll} \varphi & \\ \implies_{\mathcal{LA}(\mathbb{Z})} & (d = 0) \\ \implies_{\mathcal{EUF}} & (f(d) = f(0)) \\ \implies_{\text{Bool}} & (\text{read}(\text{write}(V, i, x), i + d) = x + 1) \\ \implies_{\mathcal{LA}(\mathbb{Z})} & (\text{read}(\text{write}(V, i, x), i) = x + 1) \\ \implies_{\mathcal{LA}(\mathbb{Z})} & \neg(\text{read}(\text{write}(V, i, x), i) = x) \\ \implies_{\mathcal{AR}} & \perp \end{array}$$

Satisfiability Modulo Theories (SMT($\mathcal{T}$)): Example

Example: $\text{SMT}(\mathcal{LA}(\mathbb{Z}) \cup \mathcal{EUF} \cup \mathcal{AR})$

$$\varphi \stackrel{\text{def}}{=} (d \geq 0) \wedge (d < 1) \wedge \\ ((f(d) = f(0)) \rightarrow (\text{read}(\text{write}(V, i, x), i + d) = x + 1))$$

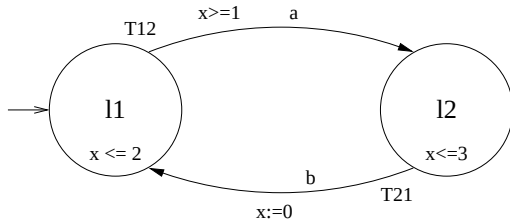
- involves arithmetical, arrays, and uninterpreted function/predicate symbols, plus Boolean operators
 - Is it consistent?
 - No:

$$\begin{array}{ll} & \varphi \\ \implies_{\mathcal{LA}(\mathbb{Z})} & (d = 0) \\ \implies_{\mathcal{EUF}} & (f(d) = f(0)) \\ \implies_{\text{Bool}} & (\text{read}(\text{write}(V, i, x), i + d) = x + 1) \\ \implies_{\mathcal{LA}(\mathbb{Z})} & (\text{read}(\text{write}(V, i, x), i) = x + 1) \\ \implies_{\mathcal{LA}(\mathbb{Z})} & \neg(\text{read}(\text{write}(V, i, x), i) = x) \\ \implies_{\mathcal{AR}} & \perp \end{array}$$

Some Motivating Applications

Interest in SMT triggered by some real-world applications

- Verification of Hybrid & Timed Systems
- Verification of RTL Circuit Designs & of Microcode
- SW Verification
- Planning with Resources
- Temporal reasoning
- Scheduling
- Compiler optimization
- ...

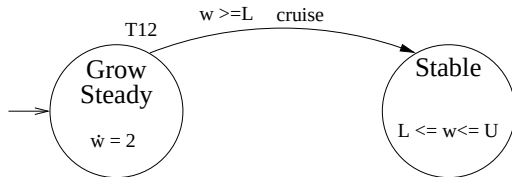


- ...

- **Timed Automata** encoded into $\mathcal{T}$ -formulas:
 - **discrete information** (locations, transitions, events) with Boolean vars.
 - **timed information** (clocks, elapsed time) with differences ($t_3 - x_3 \leq 2$), equalities ($x_4 = x_3$) and linear constraints ($t_8 - x_8 = t_2 - x_2$) on $\mathbb{Q}$

⇒ SMT on $\mathcal{DL}(\mathbb{Q})$ or $\mathcal{LA}(\mathbb{Q})$ required

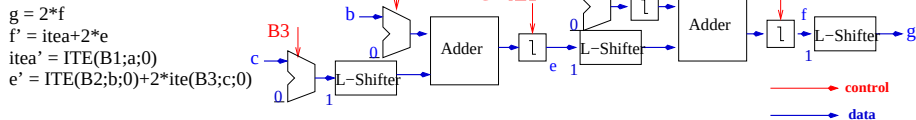
Verification of Hybrid Systems ...



- *Bounded model checking of Hybrid Systems* [5],...
- *Hybrid Automata* encoded into $\mathcal{L}$ -formulas:
 - **discrete information** (locs, trans., events) with Boolean vars.
 - **timed information** (clocks, elapsed time) with differences $(t_3 - x_3 \leq 2)$, equalities $(x_4 = x_3)$ and linear constraints $(t_8 - x_8 = t_2 - x_2)$ on $\mathbb{Q}$
 - **Evolution of Physical Variables** (e.g., speed, pressure) with linear $(\omega_4 = 2\omega_3)$ and non-linear constraints $(P_1 V_1 = 4T_1)$ on $\mathbb{Q}$
- **Undecidable** under simple hypotheses!

$\Rightarrow$ SMT on $\mathcal{DL}(\mathbb{Q})$, $\mathcal{LA}(\mathbb{Q})$ or $\mathcal{NLA}(\mathbb{Q})$ required

Verification of HW circuit designs & microcode



- SAT/SMT-based *Model Checking* & *Equiv. Checking* of RTL designs, *symbolic simulation* of μ -code [22, 19, 41]
 - *Control paths* handled by Boolean reasoning
 - *Data paths* information abstracted into theory-specific terms
 - *words* (bit-vectors, integers, $\mathcal{EUF}$ vars, ...): $\underline{a}[31 : 0]$, a
 - *word operations*: $(\mathcal{BV}, \mathcal{EUF}, \mathcal{AR}, \mathcal{LA}(\mathbb{Z}), \mathcal{NLA}(\mathbb{Z})$ operators)
 $x_{[16]}[15 : 0] = (y_{[16]}[15 : 8] :: z_{[16]}[7 : 0]) \ll w_{[8]}[3 : 0]$,
 $(a = a_L + 2^{16}a_H), (m_1 = store(m_0, l_0, v_0)), \dots$
 - Trades **heavy Boolean reasoning** ($\approx 2^{64}$ factors) with **$\mathcal{T}$ -solving**
- $\Rightarrow$ SMT on $\mathcal{BV}, \mathcal{EUF}, \mathcal{AR}$, modulo- $\mathcal{LA}(\mathbb{Z})$ [$\mathcal{NLA}(\mathbb{Z})$] required

Verification of SW systems

```

...
i = 0;
acc = init();
while (i < dim) {
    acc += V[i];
    i++;
}
...

```

$$\begin{aligned}
 & \dots \wedge \\
 & (i_0 = 0) \wedge \\
 & (acc_0 = init()) \wedge \\
 & ((i_0 < dim) \rightarrow ((acc_1 = acc_0 + read(V, i_0)) \wedge \\
 & \quad (i_1 = i_0 + 1))) \wedge \\
 & (\neg(i_0 < dim) \rightarrow ((acc_1 = acc_0) \wedge (i_1 = i_0))) \wedge \\
 & ((i_1 < dim) \rightarrow ((acc_2 = acc_1 + read(V, i_1)) \wedge \\
 & \quad (i_2 = i_1 + 1))) \wedge \\
 & (\neg(i_1 < dim) \rightarrow ((acc_2 = acc_1) \wedge (i_2 = i_1))) \wedge \\
 & \dots
 \end{aligned}$$

- Verification of SW code

- *BMC, K-induction, Check of proof obligations, interpolation-based model checking, symbolic simulation, concolic testing, ...*

$\Rightarrow$ SMT on $\mathcal{BV}$, $\mathcal{EUF}$, $\mathcal{AR}$, (modulo-) $\mathcal{LA}(\mathbb{Z})$ [$\mathcal{NLA}(\mathbb{Z})$] required

Planning with Resources [75]

- SAT-bases planning augmented with numerical constraints
- Straightforward to encode into SMT($\mathcal{LA}(\mathbb{Q})$)

Example (sketch) [75]

<i>(Deliver)</i>	$\wedge$ // goal
<i>(MaxLoad)</i>	$\wedge$ // load constraint
<i>(MaxFuel)</i>	$\wedge$ // fuel constraint
<i>(Move $\rightarrow$ MinFuel)</i>	$\wedge$ // move requires fuel
<i>(Move $\rightarrow$ Deliver)</i>	$\wedge$ // move implies delivery
<i>(GoodTrip $\rightarrow$ Deliver)</i>	$\wedge$ // a good trip requires
<i>(GoodTrip $\rightarrow$ AllLoaded)</i>	$\wedge$ // a full delivery
<i>(MaxLoad $\rightarrow$ (load ≤ 30))</i>	$\wedge$ // load limit
<i>(MaxFuel $\rightarrow$ (fuel ≤ 15))</i>	$\wedge$ // fuel limit
<i>(MinFuel $\rightarrow$ (fuel $\geq 7 + 0.5load$))</i>	$\wedge$ // fuel constraint
<i>(AllLoaded $\rightarrow$ (load = 45))</i>	//

(Disjunctive) Temporal Reasoning [73, 2]

- Temporal reasoning problems encoded as disjunctions of difference constraints

$$\begin{aligned}
 & ((x_1 - x_2 \leq 6) \quad \vee \quad (x_3 - x_4 \leq -2)) \quad \wedge \\
 & ((x_2 - x_3 \leq -2) \quad \vee \quad (x_4 - x_5 \leq 5)) \quad \wedge \\
 & ((x_2 - x_1 \leq 4) \quad \vee \quad (x_3 - x_7 \leq -6)) \quad \wedge \\
 & \dots
 \end{aligned}$$

- Straightforward to encode into SMT($\mathcal{DL}$)

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for **heavy Boolean reasoning** combined with capabilities for **reasoning in expressive decidable F.O. theories**

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine **SAT solvers** with **decision procedures** (*theory solvers*)
 - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for **heavy Boolean reasoning** combined with capabilities for **reasoning in expressive decidable F.O. theories**

- SAT alone not expressive enough
 - standard automated theorem proving inadequate (e.g., arithmetic)
 - may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine **SAT solvers** with **decision procedures** (*theory solvers*)
 - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for **heavy Boolean reasoning** combined with capabilities for **reasoning in expressive decidable F.O. theories**

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine **SAT solvers** with **decision procedures** (*theory solvers*)
 - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for **heavy Boolean reasoning** combined with capabilities for **reasoning in expressive decidable F.O. theories**

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine **SAT solvers** with **decision procedures** (*theory solvers*)
 - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

SMT and SMT solvers

Common fact about SMT problems from various applications

SMT requires capabilities for **heavy Boolean reasoning** combined with capabilities for **reasoning in expressive decidable F.O. theories**

- SAT alone not expressive enough
- standard automated theorem proving inadequate (e.g., arithmetic)
- may involve also numerical computation (e.g., simplex)

Modern SMT solvers

- combine **SAT solvers** with **decision procedures** (*theory solvers*)
 - contributions from SAT, Automated Theorem Proving (ATP), formal verification (FV) and operational research (OR)

Goal of this tutorial

Provide an overview of standard “lazy” SMT:

- foundations
- SMT-solving techniques
- beyond solving: advanced SMT functionalities
- ongoing research

Does **not** cover related approaches like:

- Eager SAT encodings
- Rewrite-based approaches

We refer to [70, 10] for an overview and references.

Notational remark (1): most/all examples in $\mathcal{LA}(\mathbb{Q})$

For better readability, in most/all the examples of this presentation we will use the theory of linear arithmetic on rational numbers ($\mathcal{LA}(\mathbb{Q})$) because of its intuitive semantics. E.g.:

$$(\neg A_1 \vee (3x_1 - 2x_2 - 3 \leq 5)) \wedge (A_2 \vee (-2x_1 + 4x_3 + 2 = 3))$$

Nevertheless, analogous examples can be built with all other theories of interest.

Notational remark (2): “constants” vs. “variables”

- Consider, e.g., the formula:

$$(\neg A_1 \vee (3x_1 - 2x_2 - 3 \leq 5)) \wedge (A_2 \vee (-2x_1 + 4x_3 + 2 = 3))$$
- How do we call A_1, A_2 ?:
 - Boolean/propositional **variables**?
 - uninterpreted **0-ary predicates**?
- How do we call x_1, x_2, x_3 ?:
 - domain **variables**?
 - uninterpreted Skolem **constants/0-ary uninterpreted functions**?
- Hint:
 - typically used in SAT, CSP and OR communities
 - typically used in logic & ATP communities

Hereafter we call A_1, A_2 “Boolean/propositional **variables**” and x_1, x_2, x_3 “domain **variables**” (logic purists, please forgive me!)

Notational remark (2): “constants” vs. “variables”

- Consider, e.g., the formula:

$$(\neg A_1 \vee (3x_1 - 2x_2 - 3 \leq 5)) \wedge (A_2 \vee (-2x_1 + 4x_3 + 2 = 3))$$
- How do we call A_1, A_2 ?:
 - Boolean/propositional **variables**?
 - uninterpreted **0-ary predicates**?
- How do we call x_1, x_2, x_3 ?:
 - domain **variables**?
 - uninterpreted Skolem **constants/0-ary uninterpreted functions**?
- Hint:
 - typically used in SAT, CSP and OR communities
 - typically used in logic & ATP communities

Hereafter we call A_1, A_2 “Boolean/propositional **variables**” and x_1, x_2, x_3 “domain **variables**” (logic purists, please forgive me!)

Notational remark (2): “constants” vs. “variables”

- Consider, e.g., the formula:

$$(\neg A_1 \vee (3x_1 - 2x_2 - 3 \leq 5)) \wedge (A_2 \vee (-2x_1 + 4x_3 + 2 = 3))$$
- How do we call A_1, A_2 ?:
 - Boolean/propositional **variables**?
 - uninterpreted **0-ary predicates**?
- How do we call x_1, x_2, x_3 ?:
 - domain **variables**?
 - uninterpreted Skolem **constants/0-ary uninterpreted functions**?
- Hint:
 - typically used in SAT, CSP and OR communities
 - typically used in logic & ATP communities

Hereafter we call A_1, A_2 “Boolean/propositional **variables**” and x_1, x_2, x_3 “domain **variables**” (logic purists, please forgive me!)

Outline

- 1 Motivations and goals
- 2 **Background: Modern SAT solving**
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Outline

- 1 Motivations and goals
- 2 **Background: Modern SAT solving**
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Basic notation & definitions

- *Boolean formula*

- $\top, \perp$ are formulas
- A *propositional atom* $A_1, A_2, A_3, \dots$ is a formula;
- if φ_1 and φ_2 are formulas, then $\neg\varphi_1, \varphi_1 \wedge \varphi_2, \varphi_1 \vee \varphi_2, \varphi_1 \rightarrow \varphi_2, \varphi_1 \leftrightarrow \varphi_2$ are formulas.
- *Literal*: a propositional atom A_i (positive literal) or its negation $\neg A_i$ (negative literal)
- Notation: if $l := \neg A_i$, then $\neg l := A_i$
- *Atoms*(φ): the set $\{A_1, \dots, A_N\}$ of atoms occurring in φ .
- a Boolean formula can be represented as a **tree** or as a **DAG**

Basic notation & definitions (cont)

- *Total truth assignment* μ for φ :
 $\mu : Atoms(\varphi) \mapsto \{\top, \perp\}$.
- *Partial Truth assignment* μ for φ :
 $\mu : \mathcal{A} \mapsto \{\top, \perp\}, \mathcal{A} \subset Atoms(\varphi)$.
- Set and formula representation of an assignment:
 - μ can be represented as a set of literals:
EX: $\{\mu(A_1) := \top, \mu(A_2) := \perp\} \implies \{A_1, \neg A_2\}$
 - μ can be represented as a formula:
EX: $\{\mu(A_1) := \top, \mu(A_2) := \perp\} \implies A_1 \wedge \neg A_2$

Basic notation & definitions (cont)

- $\mu \models \varphi$ (μ *satisfies* φ):
 - $\mu \models A_i \iff \mu(A_i) = \top$
 - $\mu \models \neg \varphi \iff \text{not } \mu \models \varphi$
 - $\mu \models \varphi_1 \wedge \varphi_2 \iff \mu \models \varphi_1 \text{ and } \mu \models \varphi_2$
 - $\mu \models \varphi_1 \vee \varphi_2 \iff \mu \models \varphi_1 \text{ or } \mu \models \varphi_2$
 - $\mu \models \varphi_1 \rightarrow \varphi_2 \iff \text{if } \mu \models \varphi_1, \text{ then } \mu \models \varphi_2$
 - $\mu \models \varphi_1 \leftrightarrow \varphi_2 \iff \mu \models \varphi_1 \text{ iff } \mu \models \varphi_2$
- φ is *satisfiable* iff $\mu \models \varphi$ for some μ
- $\varphi_1 \models \varphi_2$ (φ_1 *entails* φ_2):
 - $\varphi_1 \models \varphi_2$ iff for every μ $\mu \models \varphi_1 \implies \mu \models \varphi_2$
- $\models \varphi$ (φ *is valid*):
 - $\models \varphi$ iff for every μ $\mu \models \varphi$

Property

φ is valid $\iff \neg \varphi$ is not satisfiable

Conjunctive Normal Form (CNF)

- φ is in *Conjunctive normal form* iff it is a conjunction of disjunctions of literals:

$$\bigwedge_{i=1}^L \bigvee_{j_i=1}^{K_i} l_{j_i}$$

- the disjunctions of literals $\bigvee_{j_i=1}^{K_i} l_{j_i}$ are called *clauses*
- Easier to handle: list of lists of literals.
 $\implies$ no reasoning on the recursive structure of the formula

Labeling CNF conversion $CNF_{label}(\varphi)$ [66, 33]

- Every φ can be reduced into CNF by, e.g., applying recursively bottom-up the rules:

$$\varphi \implies \varphi[(l_i \vee l_j)|B] \wedge CNF(B \leftrightarrow (l_i \vee l_j))$$

$$\varphi \implies \varphi[(l_i \wedge l_j)|B] \wedge CNF(B \leftrightarrow (l_i \wedge l_j))$$

$$\varphi \implies \varphi[(l_i \leftrightarrow l_j)|B] \wedge CNF(B \leftrightarrow (l_i \leftrightarrow l_j))$$

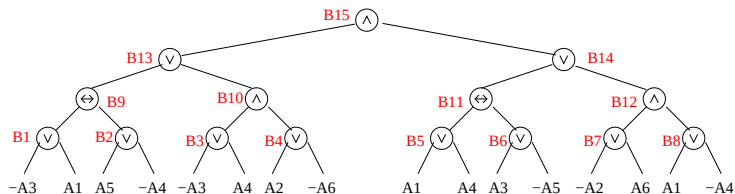
l_i, l_j being literals and B being a “new” variable.

- Worst-case **linear**.
- $Atoms(CNF_{label}(\varphi)) \supseteq Atoms(\varphi)$.
- $CNF_{label}(\varphi)$ is **equi-satisfiable** w.r.t. φ .
- More used in practice.

Labeling CNF conversion $CNF_{label}(\varphi)$ (cont.)

$CNF(B \leftrightarrow (l_i \vee l_j))$	$\iff$	$(\neg B \vee l_i \vee l_j) \wedge$ $(B \vee \neg l_i) \wedge$ $(B \vee \neg l_j)$
$CNF(B \leftrightarrow (l_i \wedge l_j))$	$\iff$	$(\neg B \vee l_i) \wedge$ $(\neg B \vee l_j) \wedge$ $(B \vee \neg l_i \vee \neg l_j)$
$CNF(B \leftrightarrow (l_i \leftrightarrow l_j))$	$\iff$	$(\neg B \vee \neg l_i \vee l_j) \wedge$ $(\neg B \vee l_i \vee \neg l_j) \wedge$ $(B \vee l_i \vee l_j) \wedge$ $(B \vee \neg l_i \vee \neg l_j)$

Labeling CNF conversion CNF_{label} – example



$$CNF(B_1 \leftrightarrow (\neg A_3 \vee A_1)) \quad \wedge \quad // \left\{ \begin{array}{l} (\neg B_1 \vee \neg A_3 \vee A_1) \\ (B_1 \vee A_3) \\ (B_1 \vee \neg A_3) \end{array} \right. \wedge \left. \right\}$$

$$\dots \quad \wedge \quad // \{ \dots \}$$

$$CNF(B_8 \leftrightarrow (A_1 \vee \neg A_4)) \quad \wedge \quad // \{ \dots \}$$

$$CNF(B_9 \leftrightarrow (B_1 \leftrightarrow B_2)) \quad \wedge \quad // \{ \dots \}$$

$$\dots \quad \wedge \quad // \{ \dots \}$$

$$CNF(B_{12} \leftrightarrow (B_7 \wedge B_8)) \quad \wedge \quad // \{ \dots \}$$

$$CNF(B_{13} \leftrightarrow (B_9 \vee B_{10})) \quad \wedge \quad // \{ \dots \}$$

$$CNF(B_{14} \leftrightarrow (B_{11} \vee B_{12})) \quad \wedge \quad // \{ \dots \}$$

$$CNF(B_{15} \leftrightarrow (B_{13} \wedge B_{14})) \quad \wedge \quad // \{ \dots \}$$

$$B_{15}$$

Resolution Rule

- Resolution of a pair of clauses with exactly one incompatible variable (*resolvent/pivot*):

$$\begin{array}{c}
 \text{common} \quad \text{resolvent} \quad C' \quad \text{common} \quad \text{resolvent} \quad C'' \\
 (l_1 \vee \dots \vee l_k \vee \textcolor{red}{l} \vee l'_{k+1} \vee \dots \vee l'_m) \quad (l_1 \vee \dots \vee l_k \vee \textcolor{red}{\neg l} \vee l''_{k+1} \vee \dots \vee l''_n) \\
 \hline
 (l_1 \vee \dots \vee l_k \vee l'_{k+1} \vee \dots \vee l'_m \vee l''_{k+1} \vee \dots \vee l''_n) \\
 \text{common} \quad C' \quad C''
 \end{array}$$

- Example:

$$\begin{array}{c}
 (A \vee B \vee \textcolor{red}{C} \vee D \vee E) \quad (A \vee B \vee \textcolor{red}{\neg C} \vee F) \\
 \hline
 (A \vee B \vee D \vee E \vee F)
 \end{array}$$

- Note: many standard inference rules subcases of resolution:

$$\frac{A \rightarrow B \quad B \rightarrow C}{A \rightarrow C} \text{ (Transit.)} \quad \frac{A \quad A \rightarrow B}{B} \text{ (M. Ponens)} \quad \frac{\neg B \quad A \rightarrow B}{\neg A} \text{ (M. Tollens)}$$

Resolution Rules [32, 31]: Unit Propagation

- *Unit resolution:*

$$\frac{\Gamma' \wedge (l) \wedge (\neg l \vee \bigvee_i l_i)}{\Gamma' \wedge (l) \wedge (\bigvee_i l_i)}$$

- *Unit subsumption:*

$$\frac{\Gamma' \wedge (l) \wedge (l \vee \bigvee_i l_i)}{\Gamma' \wedge (l)}$$

- *Unit propagation:* Unit resolution + Unit subsumption

“Deterministic” rule, applied **before** other “non-deterministic” rules!

Outline

- 1 Motivations and goals
- 2 **Background: Modern SAT solving**
 - Basics
 - **Modern CDCL SAT solving**
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Conflict-Driven Clause-Learning (CDCL) SAT solvers

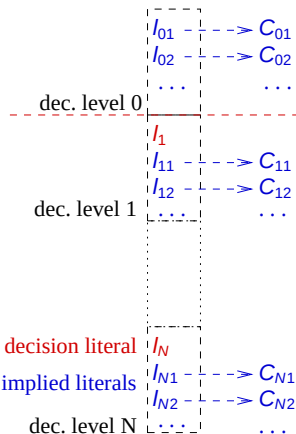
Conflict-Driven Clause-Learning (CDCL) SAT solvers [72, 59, 40]

- Evolution of Davis-Putnam-Longeman-Loveland (DPLL) [32, 31]
- non-recursive: stack-based representation of data structures
- **Perform conflict-directed backtracking (backjumping) and learning**
- efficient data structures for doing and undoing assignments (e.g., two-watched-literal scheme)
- perform search restarts
- ...

Dramatically efficient: solve industrial-derived problems with $\approx 10^7$ Boolean variables and $\approx 10^7 - 10^8$ clauses!

Stack-based representation of a truth assignment μ

- assign one truth-value at a time (add one literal to a stack representing μ)
- stack partitioned into *decision levels*:
 - one *decision literal*
 - its *implied literals*
 - each implied literal tagged with the clause causing its unit-propagation (*antecedent clause*)
- equivalent to an *implication graph*:
 - a node without incoming edges represent a *decision literal*
 - the graph contains $l_1 \xrightarrow{c} l, \dots, l_n \xrightarrow{c} l$ iff $c \stackrel{\text{def}}{=} \bigvee_{j=1}^n \neg l_j \vee l$ is the antecedent clause of l



representation of the dependencies
between literals in μ

Schema of a CDCL SAT solver

```

Function CDCL-SAT (formula:  $\varphi$ , assignment &  $\mu$ ) {
    status := preprocess( $\varphi, \mu$ );
    while (1) {
        decide_next_branch( $\varphi, \mu$ );
        while (1) {
            status := bcp( $\varphi, \mu, \eta$ );            $\eta$  is a conflict set
            if (status == Sat)
                return Sat;
            if (status == Conflict) {
                blevel := analyze_conflict( $\varphi, \mu, \eta$ );
                if (blevel == 0)
                    return Unsat;
                else backtrack(blevel,  $\varphi, \mu$ );
            }
            else break;
        }
    }
}

```

Schema of a CDCL SAT solver (2)

- **preprocess**(φ, μ) simplifies φ into an easier equisatisfiable formula (and updates μ if it is the case)
- **decide_next_branch**(φ, μ) chooses a new decision literal from φ according to some heuristic, and adds it to μ
- **bcp**(φ, μ, η) performs all deterministic assignments (unit), and updates φ and μ accordingly. If this causes a conflict, η is the subset of μ causing the conflict (conflict set).
- **analyze_conflict**(φ, μ, η) returns the “wrong-decision” level suggested by η (“0” means that a conflict exists even without branching)
- **backtrack**(**blevel**, φ, μ) undoes the branches up to **blevel**, and updates φ and μ accordingly

Example

$$C_1 : \neg A_1 \vee A_2$$

$$C_2 : \neg A_1 \vee A_3 \vee A_9$$

$$C_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$C_4 : \neg A_4 \vee A_5 \vee A_{10}$$

$$C_5 : \neg A_4 \vee A_6 \vee A_{11}$$

$$C_6 : \neg A_5 \vee \neg A_6$$

$$C_7 : A_1 \vee A_7 \vee \neg A_{12}$$

$$C_8 : A_1 \vee A_8$$

$$C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

...

Example

$$c_1 : \neg A_1 \vee A_2$$

$$c_2 : \neg A_1 \vee A_3 \vee A_9$$

$$c_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$c_4 : \neg A_4 \vee A_5 \vee A_{10}$$

$$c_5 : \neg A_4 \vee A_6 \vee A_{11}$$

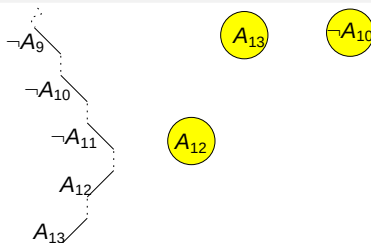
$$c_6 : \neg A_5 \vee \neg A_6$$

$$c_7 : A_1 \vee A_7 \vee \neg A_{12}$$

$$c_8 : A_1 \vee A_8$$

$$c_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \dots\}$

(Initial assignment. Note: $c_1, \dots, c_9$ inconsistent.)



Example

$$C_1 : \neg A_1 \vee A_2$$

$$C_2 : \neg A_1 \vee A_3 \vee A_9$$

$$C_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$C_4 : \neg A_4 \vee A_5 \vee A_{10}$$

$$C_5 : \neg A_4 \vee A_6 \vee A_{11}$$

$$C_6 : \neg A_5 \vee \neg A_6$$

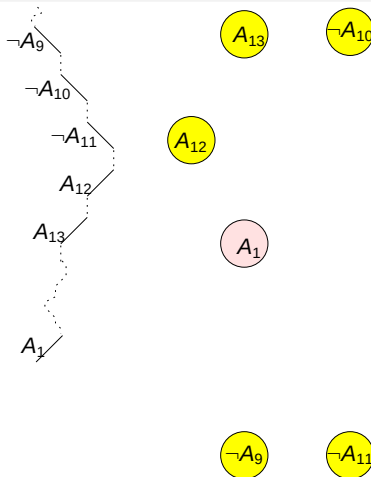
$$C_7 : A_1 \vee A_7 \vee \neg A_{12} \quad \checkmark$$

$$C_8 : A_1 \vee A_8 \quad \checkmark$$

$$C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

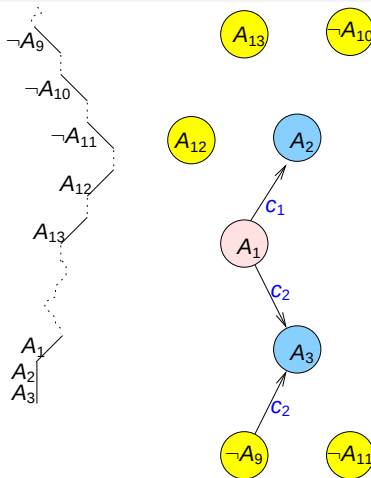
...

$\{\dots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \dots, A_1\}$
 ... (decide A_1)



Example

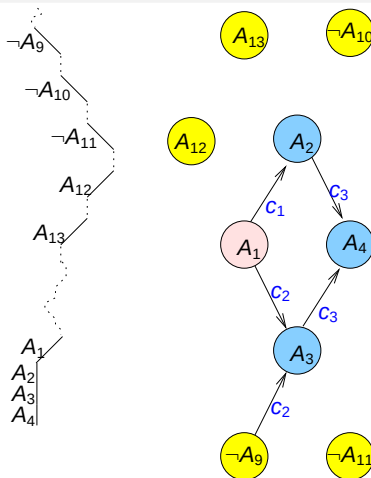
- $C_1 : \neg A_1 \vee A_2$ ✓
 $C_2 : \neg A_1 \vee A_3 \vee A_9$ ✓
 $C_3 : \neg A_2 \vee \neg A_3 \vee A_4$
 $C_4 : \neg A_4 \vee A_5 \vee A_{10}$
 $C_5 : \neg A_4 \vee A_6 \vee A_{11}$
 $C_6 : \neg A_5 \vee \neg A_6$
 $C_7 : A_1 \vee A_7 \vee \neg A_{12}$ ✓
 $C_8 : A_1 \vee A_8$ ✓
 $C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
 ...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., A_1, A_2, A_3\}$
 (unit A_2, A_3)

Example

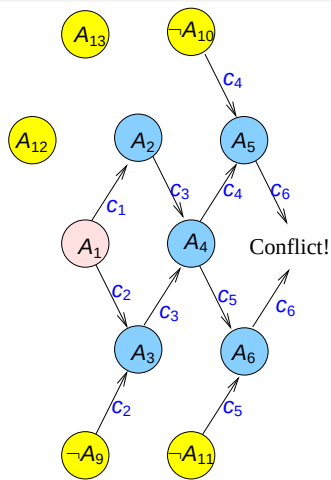
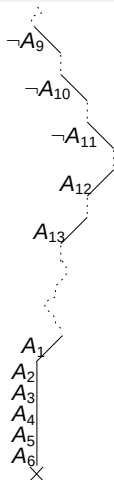
$C_1 : \neg A_1 \vee A_2 \quad \checkmark$
 $C_2 : \neg A_1 \vee A_3 \vee A_9 \quad \checkmark$
 $C_3 : \neg A_2 \vee \neg A_3 \vee A_4 \quad \checkmark$
 $C_4 : \neg A_4 \vee A_5 \vee A_{10}$
 $C_5 : \neg A_4 \vee A_6 \vee A_{11}$
 $C_6 : \neg A_5 \vee \neg A_6$
 $C_7 : A_1 \vee A_7 \vee \neg A_{12} \quad \checkmark$
 $C_8 : A_1 \vee A_8 \quad \checkmark$
 $C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
 ...



$\{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, ..., A_1, A_2, A_3, A_4\}$
 (unit A_4)

Example

- $C_1 : \neg A_1 \vee A_2$ ✓
 $C_2 : \neg A_1 \vee A_3 \vee A_9$ ✓
 $C_3 : \neg A_2 \vee \neg A_3 \vee A_4$ ✓
 $C_4 : \neg A_4 \vee A_5 \vee A_{10}$ ✓
 $C_5 : \neg A_4 \vee A_6 \vee A_{11}$ ✓
 $C_6 : \neg A_5 \vee \neg A_6$ ✗
 $C_7 : A_1 \vee A_7 \vee \neg A_{12}$ ✓
 $C_8 : A_1 \vee A_8$ ✓
 $C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$
 ...



$\{ \dots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \dots, A_1, A_2, A_3, A_4, A_5, A_6 \}$
 (unit A_5, A_6) $\Rightarrow$ conflict

State-of-the-art backjumping and learning [77]

- Idea: when a branch μ fails,
 - (i) *conflict analysis*: find the *conflict set* $\eta \subseteq \mu$ by generating the *conflict clause* $C \stackrel{\text{def}}{=} \neg\eta$ via resolution from the falsified clause
 - (ii) *learning*: add the conflict clause C to the clause set
 - (iii) *backjumping*: backtrack to the highest branching point s.t. the stack contains all-but-one literals in η , and then unit-propagate the unassigned literal on C

$\implies$ may climb up to many decision levels in the stack
- if η ($\neg C$) entirely assigned at level 0, then return UNSAT

Conflict analysis: build a conflict clause by resolution

1. $C :=$ falsified clause (*conflicting clause*)
2. repeat
 - (i) resolve the current clause C with the antecedent clause of the last unit-propagated literal l in Cuntil C verifies some given termination criteria

Conflict analysis: build a conflict clause by resolution

1. $C :=$ falsified clause (*conflicting clause*)
2. repeat
 - (i) resolve the current clause C with the antecedent clause of the last unit-propagated literal l in C
 until C verifies some given termination criteria

criterion: **decision**

...until C contains only decision literals

$$\begin{array}{c}
 \text{Conflicting cl.} \\
 \hline
 \begin{array}{c}
 \neg A_4 \vee A_6 \vee A_{11} \quad \neg A_5 \vee \neg A_6 \\
 \hline
 \neg A_4 \vee \neg A_5 \vee A_{11} \quad (A_5)
 \end{array} \\
 \hline
 \begin{array}{c}
 \neg A_4 \vee A_5 \vee A_{10} \quad \neg A_4 \vee A_{10} \vee A_{11} \\
 \hline
 \neg A_4 \vee A_{10} \vee A_{11} \quad (A_4)
 \end{array} \\
 \hline
 \begin{array}{c}
 \neg A_2 \vee \neg A_3 \vee A_4 \quad \neg A_2 \vee \neg A_3 \vee A_{10} \vee A_{11} \\
 \hline
 \neg A_2 \vee \neg A_3 \vee A_{10} \vee A_{11} \quad (A_3)
 \end{array} \\
 \hline
 \begin{array}{c}
 \neg A_1 \vee A_3 \vee A_9 \quad \neg A_2 \vee \neg A_1 \vee A_9 \vee A_{10} \vee A_{11} \\
 \hline
 \neg A_2 \vee \neg A_1 \vee A_9 \vee A_{10} \vee A_{11} \quad (A_2)
 \end{array} \\
 \hline
 \neg A_1 \vee A_2 \quad \neg A_1 \vee A_9 \vee A_{10} \vee A_{11}
 \end{array}$$

Conflict analysis: build a conflict clause by resolution

1. $C :=$ falsified clause (*conflicting clause*)
2. repeat
 - (i) resolve the current clause C with the antecedent clause of the last unit-propagated literal l in C
 until C verifies some given termination criteria

criterion: 1st UIP

... until C contains only one literal assigned at current decision level
(1st UIP)

$$\begin{array}{c}
 \text{Conflicting cl.} \\
 \overline{A_4} \vee A_5 \vee A_{10} \quad \overline{A_4} \vee A_6 \vee A_{11} \quad \overbrace{\overline{A_5} \vee \overline{A_6}}^{(A_6)} \\
 \hline
 \overline{A_4} \vee \overline{A_5} \vee A_{11} \quad (A_5) \\
 \hline
 \underbrace{\overline{A_4}}_{\text{1st UIP}} \vee A_{10} \vee A_{11}
 \end{array}$$

Conflict analysis: build a conflict clause by resolution

1. $C :=$ falsified clause (*conflicting clause*)
2. repeat
 - (i) resolve the current clause C with the antecedent clause of the last unit-propagated literal l in Cuntil C verifies some given termination criteria

Note:

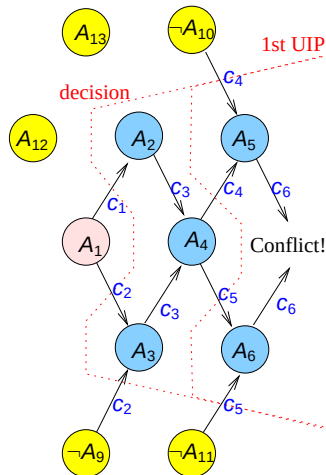
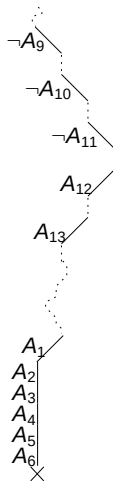
Equivalent to finding a partition in the implication graph of μ with all decision literals on one side and the conflict on the other.

Note

$\varphi \models C$, so that C can be safely added to C .

Example

- $C_1 : \neg A_1 \vee A_2$ ✓
 $C_2 : \neg A_1 \vee A_3 \vee A_9$ ✓
 $C_3 : \neg A_2 \vee \neg A_3 \vee A_4$ ✓
 $C_4 : \neg A_4 \vee A_5 \vee A_{10}$ ✓
 $C_5 : \neg A_4 \vee A_6 \vee A_{11}$ ✓
 $C_6 : \neg A_5 \vee \neg A_6$ ✗
 $C_7 : A_1 \vee A_7 \vee \neg A_{12}$ ✓
 $C_8 : A_1 \vee A_8$ ✓
 $C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$ ✓
 ...



$\Rightarrow$ Conflict set: $\{\neg A_{10}, \neg A_{11}, A_4\}$, learn $c_{10} := A_{10} \vee A_{11} \vee \neg A_4$

Example

$$C_1 : \neg A_1 \vee A_2$$

$$C_2 : \neg A_1 \vee A_3 \vee A_9$$

$$C_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$C_4 : \neg A_4 \vee A_5 \vee A_{10}$$

$$C_5 : \neg A_4 \vee A_6 \vee A_{11}$$

$$C_6 : \neg A_5 \vee \neg A_6$$

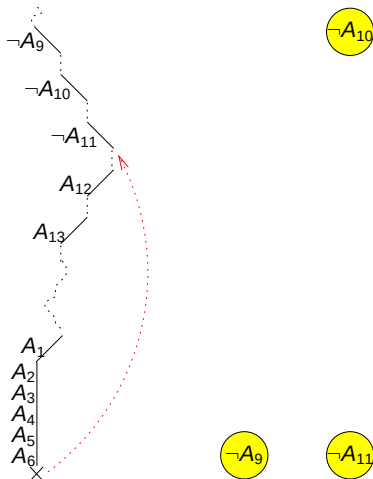
$$C_7 : A_1 \vee A_7 \vee \neg A_{12}$$

$$C_8 : A_1 \vee A_8$$

$$C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

$$C_{10} : A_{10} \vee A_{11} \vee \neg A_4$$

...



$\Rightarrow$ backtrack up to $A_{11} \Rightarrow \{..., \neg A_9, \neg A_{10}, \neg A_{11}\}$

Example

$$C_1 : \neg A_1 \vee A_2$$

$$C_2 : \neg A_1 \vee A_3 \vee A_9$$

$$C_3 : \neg A_2 \vee \neg A_3 \vee A_4$$

$$C_4 : \neg A_4 \vee A_5 \vee A_{10} \quad \checkmark$$

$$C_5 : \neg A_4 \vee A_6 \vee A_{11} \quad \checkmark$$

$$C_6 : \neg A_5 \vee \neg A_6$$

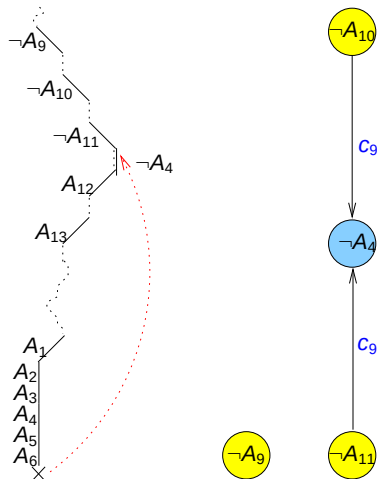
$$C_7 : A_1 \vee A_7 \vee \neg A_{12}$$

$$C_8 : A_1 \vee A_8$$

$$C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$

$$C_{10} : A_{10} \vee A_{11} \vee \neg A_4 \quad \checkmark$$

...



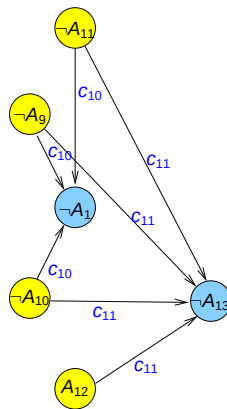
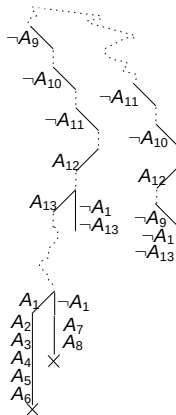
$\Rightarrow$ unit propagate $\neg A_4 \Rightarrow \{..., \neg A_9, \neg A_{10}, \neg A_{11}, A_4\}...$

Learning – example

$$C_1 : \neg A_1 \vee A_2$$
$$C_2 : \neg A_1 \vee A_3 \vee A_9$$
$$C_3 : \neg A_2 \vee \neg A_3 \vee A_4$$
$$C_4 : \neg A_4 \vee A_5 \vee A_{10}$$
$$C_5 : \neg A_4 \vee A_6 \vee A_{11}$$
$$C_6 : \neg A_5 \vee \neg A_6$$
$$C_7 : A_1 \vee A_7 \vee \neg A_{12}$$
$$C_8 : A_1 \vee A_8$$
$$C_9 : \neg A_7 \vee \neg A_8 \vee \neg A_{13}$$
$$C_{10} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$$
$$C_{11} : A_9 \vee A_{10} \vee A_{11} \vee \neg A_{12} \vee \neg A_{13} \checkmark$$

...

$\Rightarrow$ Unit: $\{\neg A_1, \neg A_{13}\}$



State-of-the-art backjumping and learning: intuitions

- *Backjumping*: allows for climbing up to many decision levels in the stack
 - intuition: “go back to the oldest decision where you’d have done something different if only you had known C ”
⇒ may avoid lots of redundant search
- *Learning*: in future branches, when all-but-one literals in η are assigned, the remaining literal is assigned to false by unit-propagation:
 - intuition: “when you’re about to repeat the mistake, do the opposite of the last step”
⇒ avoids finding the same conflict again

Drawbacks of Learning & Clause discharging

Problem with Learning

Learning can generate exponentially-many clauses

- may cause a blowup in space
- may drastically slow down BCP

A solution: clause discharging

Techniques to drop learned clauses when necessary

- according to their size
- according to their **activity**.

A clause is currently **active** if it occurs in the current implication graph (i.e., it is the antecedent clause of a literal in the current assignment).

Drawbacks of Learning & Clause discharging

Problem with Learning

Learning can generate exponentially-many clauses

- may cause a blowup in space
- may drastically slow down BCP

A solution: clause discharging

Techniques to drop learned clauses when necessary

- according to their size
- according to their **activity**.

A clause is currently **active** if it occurs in the current implication graph (i.e., it is the antecedent clause of a literal in the current assignment).

Drawbacks of Learning & Clause discharging

- Is clause-discharging safe?
- Yes, if done properly.

Property (see, e.g., [65])

In order to guarantee correctness, completeness & termination of a CDCL solver, it suffices to keep each clause until it is active.

⇒ CDCL solvers require polynomial space

Drawbacks of Learning & Clause discharging

- Is clause-discharging safe?
- Yes, if done properly.

Property (see, e.g., [65])

In order to guarantee correctness, completeness & termination of a CDCL solver, it suffices to keep each clause until it is active.

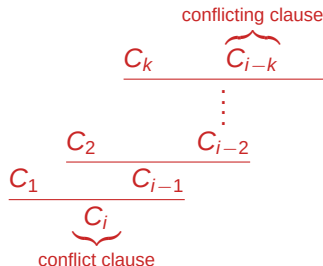
⇒ **CDCL solvers require polynomial space**

Outline

- 1 Motivations and goals
- 2 **Background: Modern SAT solving**
 - Basics
 - Modern CDCL SAT solving
 - **Other SAT functionalities**
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Building Proofs of Unsatisfiability in CDCL SAT solvers

- recall: each conflict clause C_i learned is computed from the conflicting clause C_{i-k} by backward resolving with the antecedent clause of one literal



- each resolution (sub)proof can be easily tracked:

$k \quad i-k \rightarrow i-k-1$

$\dots$

$2 \quad i-2 \rightarrow i-1$

$1 \quad i-1 \rightarrow i$

Building Proofs of Unsatisfiability in CDCL SAT solvers

- ... in particular, if φ is unsatisfiable, the last step produces “false” as conflict clause:

$$\begin{array}{ccc}
 & & \text{conflicting clause} \\
 & & \text{---} \\
 & C_k & \overbrace{C_{i-k}} \\
 & \hline
 & \vdots \\
 & C_2 & C_{i-2} \\
 & \hline
 C_1 & C_{i-1} \\
 \hline
 \perp
 \end{array}$$

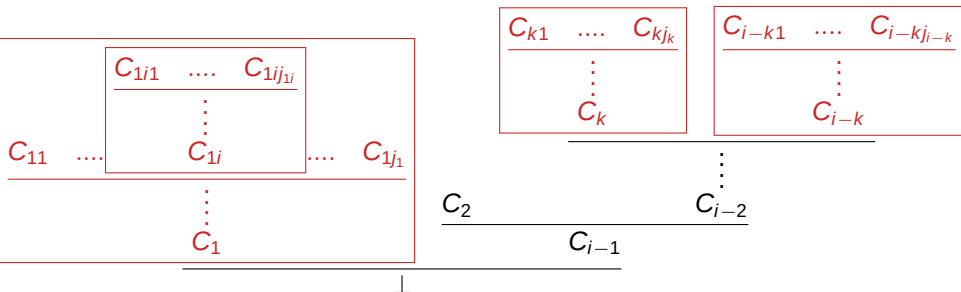
- note: $C_1 = l$, $C_{i-1} = \neg l$ for some literal l
- $C_1, \dots, C_k$, and C_{i-k} are either original or learned clauses...

Building Proofs of Unsatisfiability in CDCL SAT solvers

Starting from the previous proof of unsatisfiability, repeat recursively:

- for every **learned** leaf clause C_i , substitute C_i with the resolution proof generating it

until all leaf clauses are original clauses



⇒ we obtain a resolution proof of unsatisfiability for (a subset of) the clauses in φ

SAT under assumptions: $SAT(\varphi, \{l_1, \dots, l_n\})$

- Many SAT solvers allow for solving a CNF formula φ **under a set of assumption literals** $\mathcal{A} \stackrel{\text{def}}{=} \{l_1, \dots, l_n\}$: $SAT(\varphi, \{l_1, \dots, l_n\})$
 - $SAT(\varphi, \{l_1, \dots, l_n\})$: same result as $SAT(\varphi \wedge \bigwedge_{i=1}^n l_i)$
- Idea:
 - $l_1, \dots, l_n$ “decided” at decision level 0 before starting the search
 - if backjump to level 0 on $C \stackrel{\text{def}}{=} \neg\eta$ s.t. $\eta \subseteq \mathcal{A}$, then return UNSAT
 - if the “decision” strategy for conflict analysis is used, then η is the subset of assumptions causing the inconsistency
- incremental* calls $SAT(\varphi, \mathcal{A}_1), \dots, SAT(\varphi, \mathcal{A}_n)$ without restarting
 - stack-based interface for $\{l_1, \dots, l_n\}$
 - reuse of search (e.g. learned clauses) from call to call

SAT under assumptions: $SAT(\varphi, \{l_1, \dots, l_n\})$

- Many SAT solvers allow for solving a CNF formula φ **under a set of assumption literals** $\mathcal{A} \stackrel{\text{def}}{=} \{l_1, \dots, l_n\}$: $SAT(\varphi, \{l_1, \dots, l_n\})$
 - $SAT(\varphi, \{l_1, \dots, l_n\})$: same result as $SAT(\varphi \wedge \bigwedge_{i=1}^n l_i)$
- Idea:
 - $l_1, \dots, l_n$ “decided” at decision level 0 before starting the search
 - if backjump to level 0 on $C \stackrel{\text{def}}{=} \neg\eta$ s.t. $\eta \subseteq \mathcal{A}$, then return UNSAT
 - if the “decision” strategy for conflict analysis is used, then η **is the subset of assumptions causing the inconsistency**
- incremental* calls $SAT(\varphi, \mathcal{A}_1), \dots, SAT(\varphi, \mathcal{A}_n)$ without restarting
 - stack-based interface for $\{l_1, \dots, l_n\}$
 - reuse of search (e.g. learned clauses) from call to call

SAT under assumptions: $SAT(\varphi, \{l_1, \dots, l_n\})$

- Many SAT solvers allow for solving a CNF formula φ **under a set of assumption literals** $\mathcal{A} \stackrel{\text{def}}{=} \{l_1, \dots, l_n\}$: $SAT(\varphi, \{l_1, \dots, l_n\})$
 - $SAT(\varphi, \{l_1, \dots, l_n\})$: same result as $SAT(\varphi \wedge \bigwedge_{i=1}^n l_i)$
- Idea:
 - $l_1, \dots, l_n$ “decided” at decision level 0 before starting the search
 - if backjump to level 0 on $C \stackrel{\text{def}}{=} \neg\eta$ s.t. $\eta \subseteq \mathcal{A}$, then return UNSAT
 - if the “decision” strategy for conflict analysis is used, then η is the **subset of assumptions causing the inconsistency**
- incremental** calls $SAT(\varphi, \mathcal{A}_1), \dots, SAT(\varphi, \mathcal{A}_n)$ without restarting
 - stack-based interface for $\{l_1, \dots, l_n\}$
 - reuse of search (e.g. learned clauses) from call to call

Selection of sub-formulas

Let φ be $\bigwedge_{i=1}^n C_i$.

Idea [40, 54]

- let $S_1 \dots S_n$ be fresh Boolean atoms (*selection variables*).
 - let $\eta \stackrel{\text{def}}{=} \{S_{i_1}, \dots, S_{i_k}\} \subseteq \{S_1, \dots, S_n\}$
 - $\text{SAT}(\bigwedge_{i=1}^n (\neg S_i \vee C_i), \eta)$: same as $\text{SAT}(\bigwedge_{i=i_1}^{i_k} (C_i))$
- $\Rightarrow$ allows for “selecting” (*activating*) only a subset of the clauses in φ at each call.

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving**
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 **Efficient SMT solving**
 - **Combining SAT with Theory Solvers**
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Modern “lazy” SMT($\mathcal{T}$) solvers

A prominent “lazy” approach [44, 2, 75, 3, 8, 35]

- a *CDCL SAT solver* is used to enumerate truth assignments μ_i for (the Boolean abstraction of) the input formula φ
 - a theory-specific solver *$\mathcal{T}$ -solver* checks the $\mathcal{T}$ -consistency of the *set of $\mathcal{T}$ -literals* corresponding to each assignment
-
- Many techniques to maximize the benefits of integration [70, 10]
 - Many lazy SMT tools available
(*Barcellogic*, *CVC3*, *MathSAT*, *OpenSMT*, *SATeen*, *Yices*, *Z3*, ...)
 - Note: lazy SMT($\mathcal{T}$) approach often also referred to as “*DPLL($\mathcal{T}$)*” approach

Basic schema: example

 $\varphi =$

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

 $\varphi^p =$

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

$$B_3 \vee A_2$$

$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$

true, false

$$\mu^p = \{ \neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2 \}$$

$$\mu = \{ \neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \\ \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1) \}$$

$\Rightarrow$ inconsistent in $\mathcal{LA}(\mathbb{Q}) \Rightarrow$ backtrack

Basic schema: example

 $\varphi =$

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

 $\varphi^p =$

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

$$B_3 \vee A_2$$

$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$

true, false

$$\mu^p = \{ \neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2 \}$$

$$\mu = \{ \neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \\ \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1) \}$$

$\Rightarrow$ inconsistent in $\mathcal{LA}(\mathbb{Q}) \Rightarrow$ backtrack

Basic schema: example

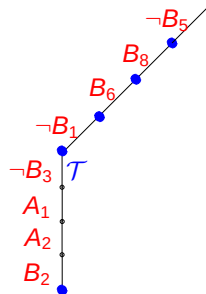
 $\varphi =$

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$
 $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$
 $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$
 $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$
 $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$
 $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$
 $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$

true, false

 $\varphi^p =$

- $\neg B_1 \vee A_1$
 $\neg A_2 \vee B_2$
 $B_3 \vee A_2$
 $\neg B_4 \vee \neg B_5 \vee \neg A_1$
 $A_1 \vee B_3$
 $B_6 \vee B_7 \vee \neg A_1$
 $A_1 \vee B_8 \vee A_2$



$$\begin{aligned}
 \mu^p &= \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\} \\
 \mu &= \{\neg(3v_1 - v_3 \leq 6), \underline{(v_3 = 3v_5 + 4)}, (v_2 - v_4 \leq 6), \\
 &\quad \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), \underline{(v_1 - v_5 \leq 1)}\}
 \end{aligned}$$

$\Rightarrow$ inconsistent in $\mathcal{LA}(\mathbb{Q}) \Rightarrow$ backtrack

Basic schema: example

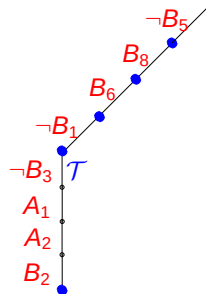
 $\varphi =$

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$
 $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$
 $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$
 $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$
 $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$
 $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$
 $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$

true, false

 $\varphi^p =$

- $\neg B_1 \vee A_1$
 $\neg A_2 \vee B_2$
 $B_3 \vee A_2$
 $\neg B_4 \vee \neg B_5 \vee \neg A_1$
 $A_1 \vee B_3$
 $B_6 \vee B_7 \vee \neg A_1$
 $A_1 \vee B_8 \vee A_2$

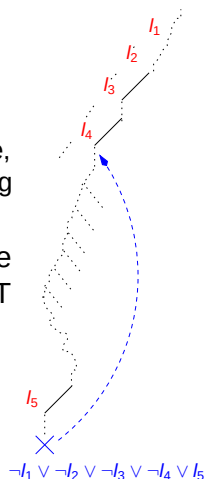


$$\begin{aligned}
 \mu^p &= \{ \neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2 \} \\
 \mu &= \{ \neg(3v_1 - v_3 \leq 6), \underline{(v_3 = 3v_5 + 4)}, (v_2 - v_4 \leq 6), \\
 &\quad \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), \underline{(v_1 - v_5 \leq 1)} \}
 \end{aligned}$$

$\Rightarrow$ inconsistent in $\mathcal{LA}(\mathbb{Q}) \Rightarrow$ backtrack

$\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning [49, 75, 3, 8, 35]

- Similar to Boolean backjumping & learning
- important property of $\mathcal{T}$ -solver:
 - **extraction of $\mathcal{T}$ -conflict sets**: if μ is $\mathcal{T}$ -unsatisfiable, then $\mathcal{T}$ -solver (μ) returns the subset η of μ causing the $\mathcal{T}$ -inconsistency of μ ($\mathcal{T}$ -conflict set)
- If so, the **$\mathcal{T}$ -conflict clause** $C := \neg\eta$ is used to drive the backjumping & learning mechanism of the SAT solver
 - $\implies$ lots of search saved
- **the less redundant is η , the more search is saved**



$\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example

 $\varphi =$

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

$$c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$$

 $\varphi^p =$

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

$$B_3 \vee A_2$$

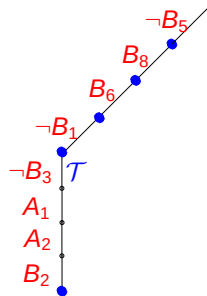
$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$

$$B_5 \vee \neg B_8 \vee \neg B_2$$



true, false

$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\}$$

$$\eta = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\}$$

$$\eta^p = \{\neg B_5, B_8, B_2\}$$

$\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example

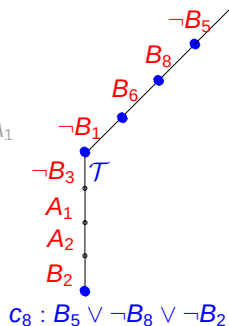
 $\varphi =$

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$
- $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$
- $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$
- $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$
- $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$
- $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$
- $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$
- $c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$

true, false

 $\varphi^p =$

- $\neg B_1 \vee A_1$
- $\neg A_2 \vee B_2$
- $B_3 \vee A_2$
- $\neg B_4 \vee \neg B_5 \vee \neg A_1$
- $A_1 \vee B_3$
- $B_6 \vee B_7 \vee \neg A_1$
- $A_1 \vee B_8 \vee A_2$
- $B_5 \vee \neg B_8 \vee \neg B_2$



$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\}$$

$$\eta = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\}$$

$$\eta^p = \{\neg B_5, B_8, B_2\}$$

$\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example

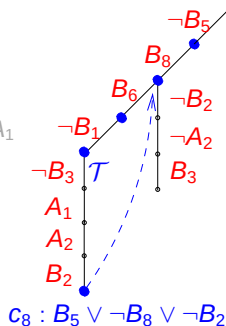
 $\varphi =$

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$
- $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$
- $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$
- $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$
- $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$
- $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$
- $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$
- $c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$

true, false

 $\varphi^p =$

- $\neg B_1 \vee A_1$
- $\neg A_2 \vee B_2$
- $B_3 \vee A_2$
- $\neg B_4 \vee \neg B_5 \vee \neg A_1$
- $A_1 \vee B_3$
- $B_6 \vee B_7 \vee \neg A_1$
- $A_1 \vee B_8 \vee A_2$
- $B_5 \vee \neg B_8 \vee \neg B_2$



$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1, \neg B_3, A_1, A_2, B_2\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2), \neg(3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1)\}$$

$$\eta = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_1 - v_5 \leq 1)\}$$

$$\eta^p = \{\neg B_5, B_8, B_2\}$$

$\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example (2)

 $\varphi =$

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

$$c'_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$$

$$c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$$

true, false

 $\varphi^p =$

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

$$B_3 \vee A_2$$

$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

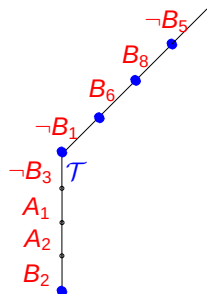
$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$

$$B_5 \vee \neg B_8 \vee B_1$$

$$B_5 \vee \neg B_8 \vee \neg B_2$$

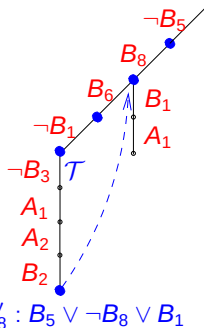


$$c_8 : B_5 \vee \neg B_8 \vee \neg B_2$$

c_8 : theory conflicting clause

$$\frac{\frac{\frac{B_5 \vee \neg B_8 \vee \neg B_2}{B_5 \vee \neg B_8 \vee \neg A_2} \quad \frac{\neg A_2 \vee B_2}{B_5 \vee \neg B_8 \vee B_3}}{B_5 \vee \neg B_8 \vee B_1} \quad (B_2) \quad (B_3) \quad \frac{B_3 \vee A_2}{B_5 \vee \neg B_8 \vee B_1} \quad (\neg A_2) \quad \frac{B_5 \vee B_1 \vee \neg B_3}{B_5 \vee \neg B_8 \vee B_1} \quad (c_T) \quad (B_3)$$

c'_8 : mixed Boolean+theory conflict clause

$\varphi =$
$$\begin{array}{ll} C_1 : & \neg(2v_2 - v_3 > 2) \vee A_1 \\ C_2 : & \neg A_2 \vee (v_1 - v_5 \leq 1) \\ C_3 : & (3v_1 - 2v_2 \leq 3) \vee A_2 \\ C_4 : & \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1 \\ C_5 : & A_1 \vee (3v_1 - 2v_2 \leq 3) \\ C_6 : & (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1 \\ C_7 : & A_1 \vee (v_3 = 3v_5 + 4) \vee A_2 \\ C'_8 : & (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots \\ C_8 : & (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots \\ & \text{true, false} \end{array}$$
$$\begin{array}{l} \neg B_1 \vee A_1 \\ \neg A_2 \vee B_2 \\ B_3 \vee A_2 \\ \neg B_4 \vee \neg B_5 \vee \neg A_1 \\ A_1 \vee B_3 \\ B_6 \vee B_7 \vee \neg A_1 \\ A_1 \vee B_8 \vee A_2 \\ B_5 \vee \neg B_8 \vee B_1 \\ B_5 \vee \neg B_8 \vee \neg B_2 \end{array}$$


c_8 : theory conflicting clause

$$\begin{array}{c}
c_8: \text{theory conflicting clause} \\
\frac{\overbrace{B_5 \vee \neg B_8 \vee \neg B_2} \quad \overbrace{\neg A_2 \vee B_2}^{c_2}}{B_5 \vee \neg B_8 \vee \neg A_2} (B_2) \quad \overbrace{B_3 \vee A_2}^{c_3} \quad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_T} \\
\frac{B_5 \vee \neg B_8 \vee \neg A_2 \quad \overbrace{B_3 \vee A_2}^{c_3}}{B_5 \vee \neg B_8 \vee B_3} (\neg A_2) \quad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_T} \\
\frac{B_5 \vee \neg B_8 \vee B_3 \quad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_T}}{B_5 \vee \neg B_8 \vee B_1} (B_3) \\
c'_8: \text{mixed Boolean+theory conflict clause}
\end{array}$$

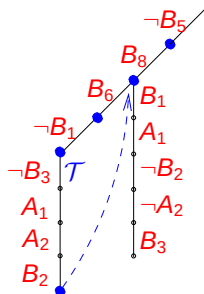
$\mathcal{T}$ -Backjumping & $\mathcal{T}$ -learning: example (2)

 $\varphi =$

- $c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$
 $c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$
 $c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$
 $c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$
 $c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$
 $c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$
 $c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$
 $c'_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$
 $c_8 : (3v_1 - v_3 \leq 6) \vee \neg(v_3 = 3v_5 + 4) \vee \dots$
true, false

 $\varphi^p =$

- $\neg B_1 \vee A_1$
 $\neg A_2 \vee B_2$
 $B_3 \vee A_2$
 $\neg B_4 \vee \neg B_5 \vee \neg A_1$
 $A_1 \vee B_3$
 $B_6 \vee B_7 \vee \neg A_1$
 $A_1 \vee B_8 \vee A_2$
 $B_5 \vee \neg B_8 \vee B_1$
 $B_5 \vee \neg B_8 \vee \neg B_2$



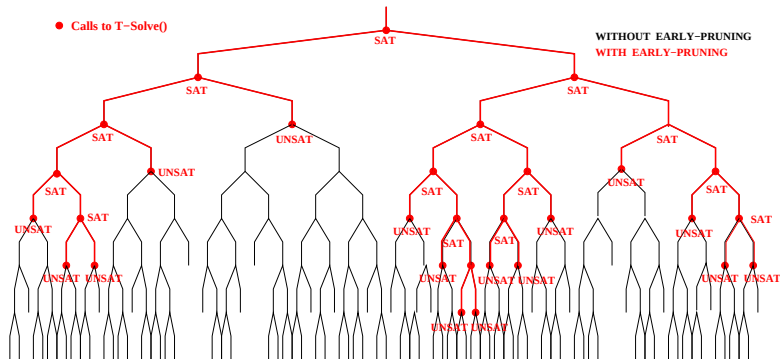
- $c'_8 : B_5 \vee \neg B_8 \vee B_1$
 $c_8 : B_5 \vee \neg B_8 \vee \neg B_2$

 c_8 : theory conflicting clause

$$\begin{array}{c}
 \overbrace{B_5 \vee \neg B_8 \vee \neg B_2}^{c_2} \quad \overbrace{\neg A_2 \vee B_2}^{c_2} \quad (B_2) \quad \overbrace{B_3 \vee A_2}^{c_3} \quad (\neg A_2) \quad \overbrace{B_5 \vee B_1 \vee \neg B_3}^{c_T} \quad (B_3) \\
 \hline
 B_5 \vee \neg B_8 \vee \neg A_2 \\
 \hline
 B_5 \vee \neg B_8 \vee B_3 \\
 \hline
 B_5 \vee \neg B_8 \vee B_1 \\
 \hline
 c'_8: \text{mixed Boolean+theory conflict clause}
 \end{array}$$

Early Pruning [44, 2, 75] I

- Introduce a $\mathcal{T}$ -satisfiability test on *intermediate assignments*: if $\mathcal{T}$ -solver returns UNSAT, the procedure backtracks.
 - benefit: prunes drastically the Boolean search
 - Drawback: possibly many useless calls to $\mathcal{T}$ -solver



Early Pruning [44, 2, 75] II

- Different strategies for interleaving Boolean search steps and $\mathcal{T}$ -solver calls
 - *Eager E.P.* [75, 11, 74, 43]): invoke $\mathcal{T}$ -solver every time a new $\mathcal{T}$ -atom is added to the assignment (unit propagations included)
 - *Selective E.P.*: Do not call $\mathcal{T}$ -solver if the have been added only literals which hardly cause any $\mathcal{T}$ -conflict with the previous assignment (e.g., Boolean literals, disequalities $(x - y \neq 3)$, $\mathcal{T}$ -literals introducing new variables $(x - z = 3)$)
 - *Weakened E.P.*: for intermediate checks only, use **weaker** but faster versions of $\mathcal{T}$ -solver (e.g., check μ on $\mathbb{R}$ rather than on $\mathbb{Z}$):
 $\{(x - y \leq 4), (z - x \leq -6), (z = y), (3x + 2y - 3z = 4)\}$

Early pruning: example

$$\begin{aligned}\varphi = & \{ \neg(2v_2 - v_3 > 2) \vee A_1 \} \wedge \\ & \{ \neg A_2 \vee (2v_1 - 4v_5 > 3) \} \wedge \\ & \{ (3v_1 - 2v_2 \leq 3) \vee A_2 \} \wedge \\ & \{ \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee (3v_1 - 2v_2 \leq 3) \} \wedge \\ & \{ (v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee (v_3 = 3v_5 + 4) \vee A_2 \}.\end{aligned}$$

$$\begin{aligned}\varphi^p = & \{ \neg B_1 \vee A_1 \} \wedge \\ & \{ \neg A_2 \vee B_2 \} \wedge \\ & \{ B_3 \vee A_2 \} \wedge \\ & \{ \neg B_4 \vee \neg B_5 \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee B_3 \} \wedge \\ & \{ B_6 \vee B_7 \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee B_8 \vee A_2 \}.\end{aligned}$$

- Suppose it is built the intermediate assignment:

$$\mu'^p = \neg B_1 \wedge \neg A_2 \wedge B_3 \wedge \neg B_5.$$

corresponding to the following set of $\mathcal{T}$ -literals

$$\mu' = \neg(2v_2 - v_3 > 2) \wedge \neg A_2 \wedge (3v_1 - 2v_2 \leq 3) \wedge \neg(3v_1 - v_3 \leq 6).$$

- If $\mathcal{T}$ -solver is invoked on μ' , then it returns UNSAT, and DPLL backtracks **without exploring any extension of μ'** .

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}\text{-solver}$ can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}\text{-solver}$ requires a *stack-based interface*

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}\text{-solver}$ can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}\text{-solver}$ requires a *stack-based interface*

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}\text{-solver}$ can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}\text{-solver}$ requires a *stack-based interface*

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2')$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2' \cup \mu_3')$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}$ -solver can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}$ -solver requires a *stack-based interface*

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}\text{-solver}$ can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}\text{-solver}$ requires a *stack-based interface*

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}\text{-solver}$ can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}\text{-solver}$ requires a *stack-based interface*

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}\text{-solver}$ can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}\text{-solver}$ requires a *stack-based interface*

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}\text{-solver}$ can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}$ -solver requires a *stack-based interface*

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}\text{-solver}$ can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}\text{-solver}$ requires a *stack-based interface*

Early pruning: remark

Incrementality & Backtrackability of $\mathcal{T}$ -solvers

- With early pruning, lots of *incremental calls to $\mathcal{T}$ -solver*:

$\mathcal{T}\text{-solver}(\mu_1)$	$\Rightarrow \text{Sat}$	Undo μ_4, μ_3, μ_2	
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3)$	$\Rightarrow \text{Sat}$	$\mathcal{T}\text{-solver}(\mu_1 \cup \mu'_2 \cup \mu'_3)$	$\Rightarrow \text{Sat}$
$\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4)$	$\Rightarrow \text{Unsat}$	...	

$\Rightarrow$ Desirable features of $\mathcal{T}$ -solvers:

- incrementality*: $\mathcal{T}\text{-solver}(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{T}\text{-solver}(\mu_1)$ without restarting from scratch
- backtrackability (resettability)*: $\mathcal{T}\text{-solver}$ can efficiently undo steps and return to a previous status on the stack

$\Rightarrow$ $\mathcal{T}$ -solver requires a **stack-based interface**

$\mathcal{T}$ -Propagation [2, 3, 43]

- strictly related to early pruning
- important property of $\mathcal{T}$ -solver:
 - **$\mathcal{T}$ -deduction**: *when a partial assignment μ is $\mathcal{T}$ -satisfiable, $\mathcal{T}$ -solver may be able to return also an assignment η to some unassigned atom occurring in φ s.t. $\mu \models_{\mathcal{T}} \eta$.*
- If so:
 - the literal η is then unit-propagated;
 - optionally, a **$\mathcal{T}$ -deduction clause** $C := \neg\mu' \vee \eta$ can be learned, μ' being the subset of μ which caused the deduction ($\mu' \models_{\mathcal{T}} \eta$)

$\Rightarrow$ may prune drastically the search

Both $\mathcal{T}$ -deduction clauses and $\mathcal{T}$ -conflict clauses are called **$\mathcal{T}$ -lemmas** since they are valid in $\mathcal{T}$

$\mathcal{T}$ -Propagation [2, 3, 43]

- strictly related to early pruning
- important property of $\mathcal{T}$ -solver:
 - **$\mathcal{T}$ -deduction**: *when a partial assignment μ is $\mathcal{T}$ -satisfiable, $\mathcal{T}$ -solver may be able to return also an assignment η to some unassigned atom occurring in φ s.t. $\mu \models_{\mathcal{T}} \eta$.*
- If so:
 - the literal η is then unit-propagated;
 - optionally, a **$\mathcal{T}$ -deduction clause** $C := \neg\mu' \vee \eta$ can be learned, μ' being the subset of μ which caused the deduction ($\mu' \models_{\mathcal{T}} \eta$)

$\Rightarrow$ may prune drastically the search

Both $\mathcal{T}$ -deduction clauses and $\mathcal{T}$ -conflict clauses are called **$\mathcal{T}$ -lemmas** since they are valid in $\mathcal{T}$

$\mathcal{T}$ -propagation: example

 $\varphi =$

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

 $\varphi^p =$

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

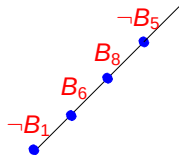
$$B_3 \vee A_2$$

$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$



true, false

$\Rightarrow$ propagate $\neg B_3$ [and learn the deduction clause $B_5 \vee B_1 \vee \neg B_3$]

$\mathcal{T}$ -propagation: example

 $\varphi =$

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

 $\varphi^p =$

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

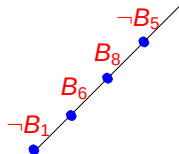
$$B_3 \vee A_2$$

$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$



true, false

$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2)\}$$

$$\models_{\mathcal{LA}(\mathbb{Q})} \underbrace{\neg(3v_1 - 2v_2 \leq 3)}_{\neg B_3}$$

$\Rightarrow$ propagate $\neg B_3$ [and learn the deduction clause $B_5 \vee B_1 \vee \neg B_3$]

$\mathcal{T}$ -propagation: example

 $\varphi =$

$$c_1 : \neg(2v_2 - v_3 > 2) \vee A_1$$

$$c_2 : \neg A_2 \vee (v_1 - v_5 \leq 1)$$

$$c_3 : (3v_1 - 2v_2 \leq 3) \vee A_2$$

$$c_4 : \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1$$

$$c_5 : A_1 \vee (3v_1 - 2v_2 \leq 3)$$

$$c_6 : (v_2 - v_4 \leq 6) \vee (v_5 = 5 - 3v_4) \vee \neg A_1$$

$$c_7 : A_1 \vee (v_3 = 3v_5 + 4) \vee A_2$$

 $\varphi^p =$

$$\neg B_1 \vee A_1$$

$$\neg A_2 \vee B_2$$

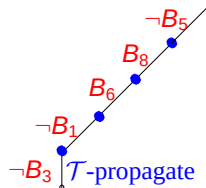
$$B_3 \vee A_2$$

$$\neg B_4 \vee \neg B_5 \vee \neg A_1$$

$$A_1 \vee B_3$$

$$B_6 \vee B_7 \vee \neg A_1$$

$$A_1 \vee B_8 \vee A_2$$



true, false

$$\mu^p = \{\neg B_5, B_8, B_6, \neg B_1\}$$

$$\mu = \{\neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4), (v_2 - v_4 \leq 6), \neg(2v_2 - v_3 > 2)\}$$

$$\models_{\mathcal{LA}(\mathbb{Q})} \underbrace{\neg(3v_1 - 2v_2 \leq 3)}_{\neg B_3}$$

$\Rightarrow$ propagate $\neg B_3$ [and learn the deduction clause $B_5 \vee B_1 \vee \neg B_3$]

Pure-literal filtering [75, 3, 14]

Property

If we have non-Boolean $\mathcal{T}$ -atoms occurring only positively [negatively] **in the original formula** φ (learned clauses are not considered), we can drop every negative [positive] occurrence of them from the assignment to be checked by $\mathcal{T}$ -solver (and from the $\mathcal{T}$ -deducible ones).

- increases the chances of finding a model
- reduces the effort for the $\mathcal{T}$ -solver
- eliminates unnecessary “nasty” negated literals (e.g. negative equalities like $\neg(3v_1 - 9v_2 = 3)$ in $\mathcal{LA}(\mathbb{Z})$ force splitting: $(3v_1 - 9v_2 > 3) \vee (3v_1 - 9v_2 < 3)$).
- may weaken the effect of early pruning.

Pure literal filtering: example

$$\begin{aligned} \varphi = & \{ \neg(2v_2 - v_3 > 2) \vee A_1 \} \wedge \\ & \{ \neg A_2 \vee (2v_1 - 4v_5 > 3) \} \wedge \\ & \{ (3v_1 - 2v_2 \leq 3) \vee A_2 \} \wedge \\ & \{ \neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee (3v_1 - 2v_2 \leq 3) \} \wedge \\ & \{ (v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1 \} \wedge \\ & \{ A_1 \vee (v_3 = 3v_5 + 4) \vee A_2 \} \wedge \\ & \{ (2v_2 - v_3 > 2) \vee \neg(3v_1 - 2v_2 \leq 3) \vee (3v_1 - v_3 \leq 6) \} \text{ learned} \end{aligned}$$

$$\mu' = \{ \neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \leq 3), \neg A_1, (v_3 = 3v_5 + 4), (3v_1 - v_3 \leq 6) \}.$$

$\Rightarrow$ Sat: $v_1 = v_2 = v_3 = 0, v_5 = -4/3$ is a solution

N.B. $(3v_1 - v_3 \leq 6)$ “filtered out” from μ' because it occurs only negatively in the original formula φ

Preprocessing atoms [44, 49, 4]

Source of inefficiency: **semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]**

- ⇒ they may be assigned different [resp. identical] truth values.
- ⇒ lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.

Preprocessing atoms [44, 49, 4]

Source of inefficiency: **semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]**

⇒ they may be assigned different [resp. identical] truth values.

⇒ lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.

Preprocessing atoms [44, 49, 4]

Source of inefficiency: **semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]**

- ⇒ they may be assigned different [resp. identical] truth values.
- ⇒ lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.

Preprocessing atoms [44, 49, 4]

Source of inefficiency: **semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other]**

- ⇒ they may be assigned different [resp. identical] truth values.
- ⇒ lots of redundant unsatisfiable assignment generated

Solution

Rewrite a priori trivially-equivalent atoms/literals into the same atom/literal.

Preprocessing atoms (cont.)

- *Sorting*: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \implies (v_1 + v_2 - v_3 \leq 1)$;
- *Rewriting dual operators*:
 $(v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg(v_1 < v_2)$
- *Exploiting associativity*:
 $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1 \implies (v_1 + v_2 + v_3 = 1)$;
- *Factoring* $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \implies (0.25v_1 + 0.5v_2 \leq 1.0)$;
- *Exploiting properties of $\mathbb{T}$* :
 $(v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Preprocessing atoms (cont.)

- *Sorting*: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \implies (v_1 + v_2 - v_3 \leq 1)$;
- *Rewriting dual operators*:
 $(v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg(v_1 < v_2)$
- *Exploiting associativity*:
 $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1 \implies (v_1 + v_2 + v_3 = 1)$;
- *Factoring* $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \implies (0.25v_1 + 0.5v_2 \leq 1.0)$;
- *Exploiting properties of $\mathbb{T}$* :
 $(v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Preprocessing atoms (cont.)

- *Sorting*: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \implies (v_1 + v_2 - v_3 \leq 1)$;
- *Rewriting dual operators*:
 $(v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg(v_1 < v_2)$
- *Exploiting associativity*:
 $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1 \implies (v_1 + v_2 + v_3 = 1)$;
- *Factoring* $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \implies (0.25v_1 + 0.5v_2 \leq 1.0)$;
- *Exploiting properties of $\mathbb{T}$* :
 $(v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Preprocessing atoms (cont.)

- *Sorting*: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \implies (v_1 + v_2 - v_3 \leq 1)$;
- *Rewriting dual operators*:
 $(v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg(v_1 < v_2)$
- *Exploiting associativity*:
 $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1 \implies (v_1 + v_2 + v_3 = 1)$;
- *Factoring* $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \implies (0.25v_1 + 0.5v_2 \leq 1.0)$;
- *Exploiting properties of $\mathbb{T}$* :
 $(v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Preprocessing atoms (cont.)

- *Sorting*: $(v_1 + v_2 \leq v_3 + 1), (v_2 + v_1 \leq v_3 + 1), (v_1 + v_2 - 1 \leq v_3) \implies (v_1 + v_2 - v_3 \leq 1)$;
- *Rewriting dual operators*:
 $(v_1 < v_2), (v_1 \geq v_2) \implies (v_1 < v_2), \neg(v_1 < v_2)$
- *Exploiting associativity*:
 $(v_1 + (v_2 + v_3) = 1), ((v_1 + v_2) + v_3) = 1 \implies (v_1 + v_2 + v_3 = 1)$;
- *Factoring* $(v_1 + 2.0v_2 \leq 4.0), (-2.0v_1 - 4.0v_2 \geq -8.0), \implies (0.25v_1 + 0.5v_2 \leq 1.0)$;
- *Exploiting properties of $\mathbb{T}$* :
 $(v_1 \leq 3), (v_1 < 4) \implies (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Static Learning [2, 4]

- Often possible to quickly detect a priori short and “obviously inconsistent” pairs or triplets of literals occurring in φ .
 - mutual exclusion $\{x = 0, x = 1\}$,
 - congruence $\{(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))\}$,
 - transitivity $\{(x - y = 2), (y - z \leq 4), \neg(x - z \leq 7)\}$,
 - substitution $\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$
 - ...
 - Preprocessing step: detect these literals and add blocking clauses to the input formula:
(e.g., $\neg(x = 0) \vee \neg(x = 1)$)
- ⇒ No assignment including one such group of literals is ever generated: as soon as all but one literals are assigned, the remaining one is immediately assigned false by unit-propagation.

Static Learning [2, 4]

- Often possible to quickly detect a priori short and “obviously inconsistent” pairs or triplets of literals occurring in φ .
 - mutual exclusion $\{x = 0, x = 1\}$,
 - congruence $\{(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))\}$,
 - transitivity $\{(x - y = 2), (y - z \leq 4), \neg(x - z \leq 7)\}$,
 - substitution $\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$
 - ...
- Preprocessing step: detect these literals and add blocking clauses to the input formula:
(e.g., $\neg(x = 0) \vee \neg(x = 1)$)

⇒ No assignment including one such group of literals is ever generated: as soon as all but one literals are assigned, the remaining one is immediately assigned false by unit-propagation.

Static Learning [2, 4]

- Often possible to quickly detect a priori short and “obviously inconsistent” pairs or triplets of literals occurring in φ .
 - mutual exclusion $\{x = 0, x = 1\}$,
 - congruence $\{(x_1 = y_1), (x_2 = y_2), \neg(f(x_1, x_2) = f(y_1, y_2))\}$,
 - transitivity $\{(x - y = 2), (y - z \leq 4), \neg(x - z \leq 7)\}$,
 - substitution $\{(x = y), (2x - 3z \leq 3), \neg(2y - 3z \leq 3)\}$
 - ...
- Preprocessing step: detect these literals and add blocking clauses to the input formula:
(e.g., $\neg(x = 0) \vee \neg(x = 1)$)

⇒ **No assignment including one such group of literals is ever generated:** as soon as all but one literals are assigned, the remaining one is immediately assigned false by unit-propagation.

Other optimization techniques

- *$\mathcal{T}$ -deduced-literal filtering*
- *Ghost-literal filtering*
- *$\mathcal{T}$ -solver layering*
- *$\mathcal{T}$ -solver clustering*
- ...

(see [70, 10] for an overview)

Other SAT-solving techniques for SMT?

Frequently-asked question:

Are CDCL SAT solvers the only suitable Boolean Engines for SMT?

Some previous attempts:

- Ordered Binary Decision Diagrams (OBDDs) [76, 58, 1]
- Stochastic Local Search [48]

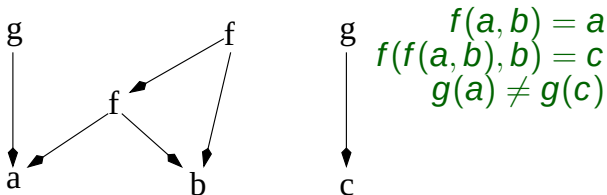
CDCL based currently much more efficient.

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 **Efficient SMT solving**
 - Combining SAT with Theory Solvers
 - **Theory Solvers for theories of interest**
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

$\mathcal{T}$ -solvers for Equality and Uninterpreted Functions ($\mathcal{EUF}$)

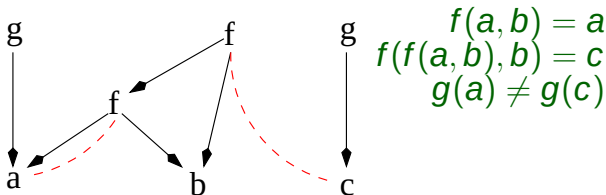
- $\mathcal{EUF}$ polynomial: $O(n \cdot \log(n))$
- use a congruence closure data structures (**E-Graphs**) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- Idea (sketch):
 - if $(t = s)$, then merge the eq. classes of t and s
 - if $\forall i \in 1 \dots k, t_i$ and s_i pairwise belong to the same eq. classes, then merge the the eq. classes of $f(t_1, \dots, t_k)$ and $f(s_1, \dots, s_k)$
 - if $(t \neq s)$ and t and s belong to the same eq. class, then conflict



Example borrowed from [38].

$\mathcal{T}$ -solvers for Equality and Uninterpreted Functions ($\mathcal{EUF}$)

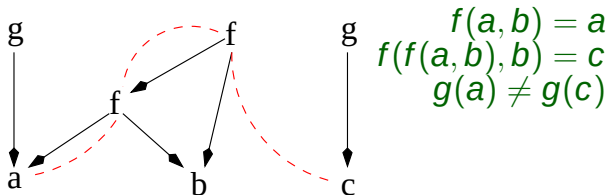
- $\mathcal{EUF}$ polynomial: $O(n \cdot \log(n))$
- use a congruence closure data structures (**E-Graphs**) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- Idea (sketch):
 - if $(t = s)$, then merge the eq. classes of t and s
 - if $\forall i \in 1 \dots k, t_i$ and s_i pairwise belong to the same eq. classes, then merge the the eq. classes of $f(t_1, \dots, t_k)$ and $f(s_1, \dots, s_k)$
 - if $(t \neq s)$ and t and s belong to the same eq. class, then conflict



Example borrowed from [38].

$\mathcal{T}$ -solvers for Equality and Uninterpreted Functions ($\mathcal{EUF}$)

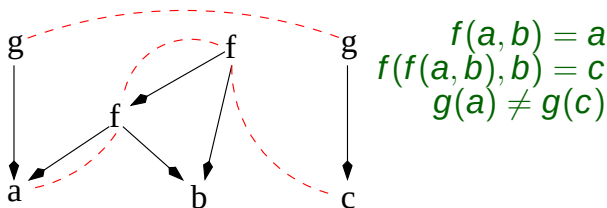
- $\mathcal{EUF}$ polynomial: $O(n \cdot \log(n))$
- use a congruence closure data structures (**E-Graphs**) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- Idea (sketch):
 - if $(t = s)$, then merge the eq. classes of t and s
 - if $\forall i \in 1 \dots k, t_i$ and s_i pairwise belong to the same eq. classes, then merge the the eq. classes of $f(t_1, \dots, t_k)$ and $f(s_1, \dots, s_k)$
 - if $(t \neq s)$ and t and s belong to the same eq. class, then conflict



Example borrowed from [38].

$\mathcal{T}$ -solvers for Equality and Uninterpreted Functions ($\mathcal{EUF}$)

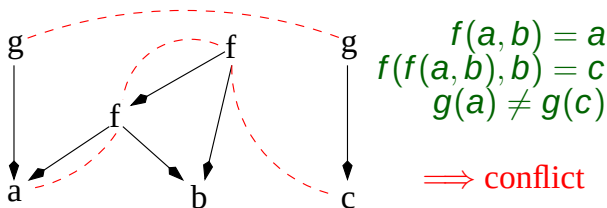
- $\mathcal{EUF}$ polynomial: $O(n \cdot \log(n))$
- use a congruence closure data structures (**E-Graphs**) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- Idea (sketch):
 - if $(t = s)$, then merge the eq. classes of t and s
 - if $\forall i \in 1 \dots k, t_i$ and s_i pairwise belong to the same eq. classes, then merge the the eq. classes of $f(t_1, \dots, t_k)$ and $f(s_1, \dots, s_k)$
 - if $(t \neq s)$ and t and s belong to the same eq. class, then conflict



Example borrowed from [38].

$\mathcal{T}$ -solvers for Equality and Uninterpreted Functions ($\mathcal{EUF}$)

- $\mathcal{EUF}$ polynomial: $O(n \cdot \log(n))$
- use a congruence closure data structures (**E-Graphs**) [38, 61, 34], based on the Union-Find data-structure for equivalence classes
- Idea (sketch):
 - if $(t = s)$, then merge the eq. classes of t and s
 - if $\forall i \in 1 \dots k, t_i$ and s_i pairwise belong to the same eq. classes, then merge the the eq. classes of $f(t_1, \dots, t_k)$ and $f(s_1, \dots, s_k)$
 - if $(t \neq s)$ and t and s belong to the same eq. class, then conflict

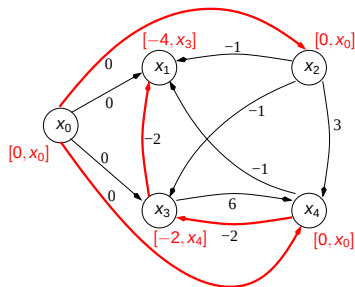


Example borrowed from [38].

$\mathcal{T}$ -solvers for Difference logic ($\mathcal{DL}$)

- $\mathcal{DL}$ polynomial: $O(\#vars \cdot \#constraints)$
- variants of the Bellman-Ford shortest-path algorithm: a negative cycle reveals a conflict [62, 30]
- Ex:

$$\{(x_1 - x_2 \leq -1), (x_1 - x_4 \leq -1), (x_1 - x_3 \leq -2), \\ (x_3 - x_1 \leq -1), (x_3 - x_2 \leq -1), (x_4 - x_2 \leq 3), (x_4 - x_3 \leq 6)\}$$

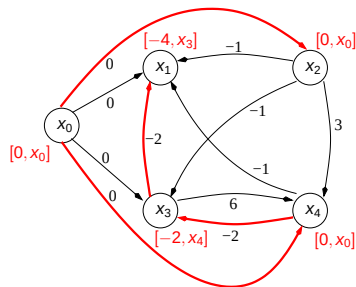


$\Rightarrow$ Sat

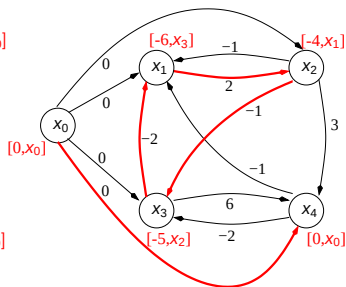
$\mathcal{T}$ -solvers for Difference logic ($\mathcal{DL}$)

- $\mathcal{DL}$ polynomial: $O(\#vars \cdot \#constraints)$
- variants of the Bellman-Ford shortest-path algorithm: a negative cycle reveals a conflict [62, 30]
- Ex:

$$\{(x_1 - x_2 \leq -1), (x_1 - x_4 \leq -1), (x_1 - x_3 \leq -2), (x_2 - x_1 \leq 2), (x_3 - x_1 \leq -1), (x_3 - x_2 \leq -1), (x_4 - x_2 \leq 3), (x_4 - x_3 \leq 6)\}$$



$\Rightarrow$ Sat



$\Rightarrow$ Unsat

$\mathcal{T}$ -solvers for Linear arithmetic over the rationals ($\mathcal{LA}(\mathbb{Q})$)

- EX: $\{(s_1 - s_2 \leq 5.2), (s_1 = s_0 + 3.4 \cdot t - 3.4 \cdot t_0), \neg(s_1 = s_0)\}$
- $\mathcal{LA}(\mathbb{Q})$ polynomial
- variants of the simplex LP algorithm [39]
- [39] allows for detecting conflict sets & performing $\mathcal{T}$ -propagation
- strict inequalities $t < 0$ rewritten as $t + \epsilon \leq 0$, ϵ treated symbolically

$$\begin{matrix} \mathcal{B} \\ \left[\begin{array}{c} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_N \end{array} \right] \end{matrix} = \begin{matrix} \left[\begin{array}{ccc} \dots A_{1j} \dots \\ \vdots \\ A_{i1} \dots A_{ij} \dots A_{iM} \\ \vdots \\ \dots A_{Nj} \dots \end{array} \right] \end{matrix} \begin{matrix} \mathcal{N} \\ \left[\begin{array}{c} x_{N+1} \\ \vdots \\ x_j \\ \vdots \\ x_{N+M} \end{array} \right] \end{matrix} ;$$

Invariant: $\beta(x_j) \in [l_j, u_j] \forall x_j \in \mathcal{N}$

Remark: infinite precision arithmetic

In order to avoid incorrect results due to numerical errors and to overflows, all $\mathcal{T}$ -solvers for $\mathcal{LA}(\mathbb{Q})$, $\mathcal{LA}(\mathbb{Z})$ and their subtheories which are based on numerical algorithms must be implemented on top of infinite-precision-arithmetic software packages.

$\mathcal{T}$ -solvers for Linear arithmetic over the integers ($\mathcal{LA}(\mathbb{Z})$)

- EX: $\{(x := x_l + 2^{16}x_h), (x \geq 0), (x \leq 2^{16} - 1)\}$
- $\mathcal{LA}(\mathbb{Z})$ NP-complete
- combination of many techniques: simplex, branch&bound, cutting planes, ... [39, 46]

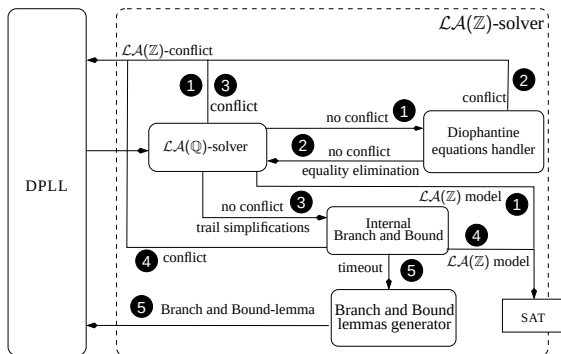


Figure courtesy of A. Griggio [46]

$\mathcal{T}$ -solvers for Arrays ($\mathcal{AR}$)

- EX: $(write(A, i, v) = write(B, i, w)) \wedge \neg(v = w)$
- NP-complete
- congruence closure ($\mathcal{EUF}$) plus on-the-fly instantiation of array's axioms:

$$\forall a. \forall i. \forall e. (read(write(a, i, e), i) = e), \quad (1)$$

$$\forall a. \forall i. \forall j. \forall e. ((i \neq j) \rightarrow read(write(a, i, e), j) = read(a, j)), \quad (2)$$

$$\forall a. \forall b. (\forall i. (read(a, i) = read(b, i)) \rightarrow (a = b)). \quad (3)$$

- EX:

Input : $(write(A, i, v) = write(B, i, w)) \wedge \neg(v = w)$

inst. (1) : $(read(write(A, i, v), i) = v)$

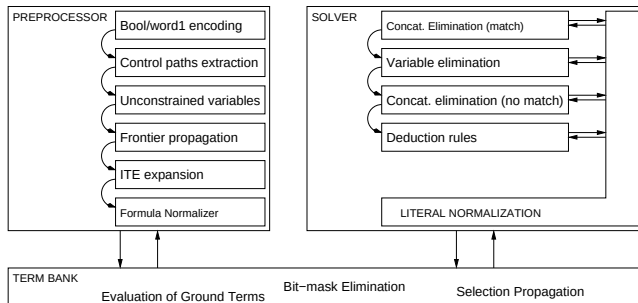
$(read(write(B, i, w), i) = w)$

$\models_{\mathcal{EUF}} (v = w)$

$\models_{Bool} \perp$

$\mathcal{T}$ -solvers for Bit vectors ($\mathcal{BV}$)

- EX: $\{(x_{[16]}[15 : 0] = (y_{[16]}[15 : 8] :: z_{[16]}[7 : 0]) << w_{[16]}[3 : 0]), \dots\}$
- NP-complete
- involve complex word-level operations: word partition/concat, modulo- 2^N arithmetic, shifts, bitwise-operations, multiplexers, ...
- combination of rewriting & simplification techniques with either:
 - final encoding into SAT [22, 42, 18, 41]
 - final encoding into $\mathcal{LA}(\mathbb{Z})$ [16, 19]



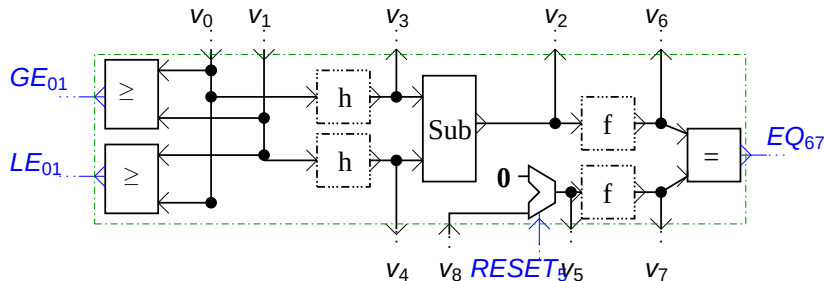
Example borrowed from [19]

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 **Efficient SMT solving**
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - **SMT for combinations of theories**
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

SMT for combined theories: $SMT(\bigcup_i \mathcal{T}_i)$

Problem: Many problems can be expressed as SMT problems only in combination of theories $\bigcup_i \mathcal{T}_i$ — $SMT(\bigcup_i \mathcal{T}_i)$



$\mathcal{LA}(\mathbb{Z}) :$ $(GE_{01} \leftrightarrow (v_0 \geq v_1)) \wedge (LE_{01} \leftrightarrow (v_0 \leq v_1)) \wedge$
 $\mathcal{EUF} :$ $(v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge$
 $\mathcal{LA}(\mathbb{Z}) :$ $(v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge$
 $\mathcal{EUF} \text{ or } \mathcal{LA}(\mathbb{Z}) :$ $(\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge$
 $\mathcal{EUF} :$ $(v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge$
 $\mathcal{EUF} \text{ or } \mathcal{LA}(\mathbb{Z}) :$ $(EQ_{67} \leftrightarrow (v_6 = v_7)) \wedge \dots$

SMT($\bigcup_i \mathcal{T}_i$) via “classic” Nelson-Oppen

Main idea

Combine two or more $\mathcal{T}_i$ -solvers into one $(\bigcup_i \mathcal{T}_i)$ -solver via *Nelson-Oppen/Shostak (N.O.) combination procedure* [60, 71]

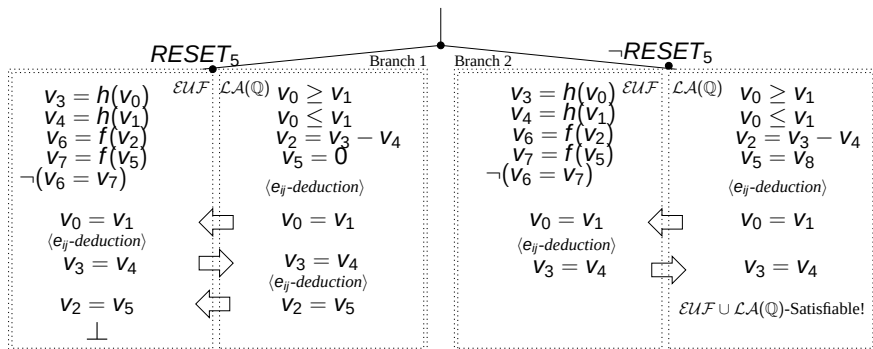
- based on the deduction and exchange of equalities between shared variables/terms (*interface equalities*, e_{ij} s)
- important improvements and evolutions [68, 7, 38]
- drawbacks [20, 21]:
 - require (possibly expensive) deduction capabilities from $\mathcal{T}_i$ -solvers
 - [with non-convex theories] case-splits forced by the deduction of disjunctions of e_{ij} 's
 - generate (typically long) $(\bigcup_i \mathcal{T}_i)$ -lemmas, without interface equalities
 $\implies$ no backjumping & learning from e_{ij} -reasoning

N.O.: example

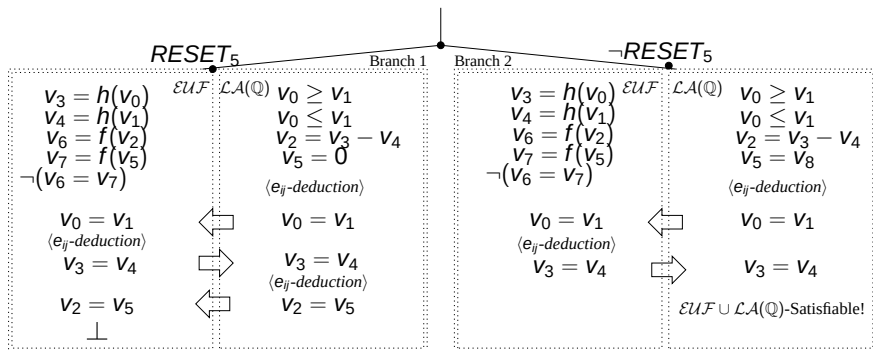
$$\mathcal{EUF} : (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge$$

$$\mathcal{LA}(\mathbb{Q}) : (v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (\text{RESET}_5 \rightarrow (v_5 = 0)) \wedge$$

$$\text{Both} : (\neg \text{RESET}_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).$$



N.O.: example (cont.)



EUF-conflict :

LA(Q)-deduction :

EUF-deduction :

LA(Q)-deduction :

$\Rightarrow$

EUF ∪ LA(Q)-conflict :

$((v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \neg(v_6 = v_7) \wedge (v_2 = v_5)) \rightarrow \perp$

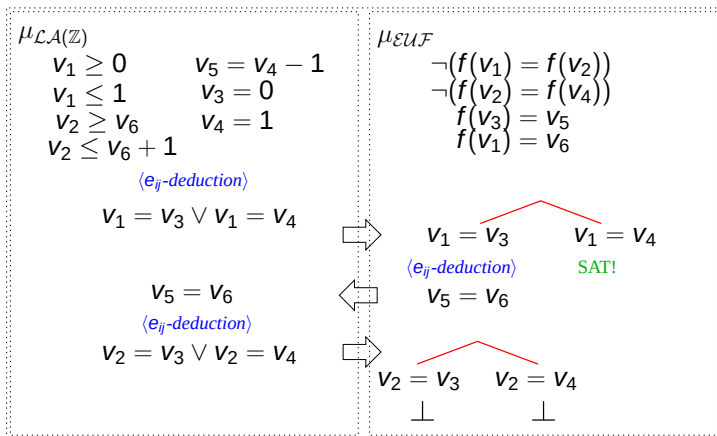
$((v_2 = v_3 - v_4) \wedge (v_5 = 0) \wedge (v_3 = v_4)) \rightarrow (v_2 = v_5)$

$((v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_0 = v_1)) \rightarrow (v_3 = v_4)$

$((v_0 \geq v_1) \wedge (v_0 \leq v_1)) \rightarrow (v_0 = v_1)$

$((v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge \neg(v_6 = v_7) \wedge (v_2 = v_3 - v_4) \wedge (v_5 = 0) \wedge (v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_0 \geq v_1)) \rightarrow \perp$

N.O.: example (non-convex theory)



SMT($\bigcup_i \mathcal{T}_i$) via Delayed Theory Combination (DTC)

Main idea

Delegate to the CDCL SAT solver part/most of the (possibly very expensive) reasoning effort on interface equalities previously due to the $\mathcal{T}_i$ -solvers (e_{ij} -deduction, case-split). [12, 13, 21]

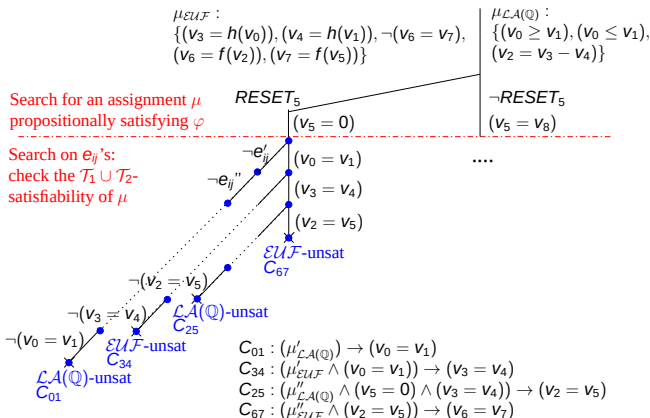
- based on Boolean reasoning on interface equalities via CDCL (plus $\mathcal{T}$ -propagation)
- important improvements and evolutions [36, 9]
- feature wrt N.O. [20, 21]
 - do not require (possibly expensive) deduction capabilities from $\mathcal{T}_i$ -solvers
 - with non-convex theories, case-splits on e_{ij} 's handled by SAT
 - generate $\mathcal{T}_i$ -lemmas with interface equalities
 $\implies$ backjumping & learning from e_{ij} -reasoning

DTC: example

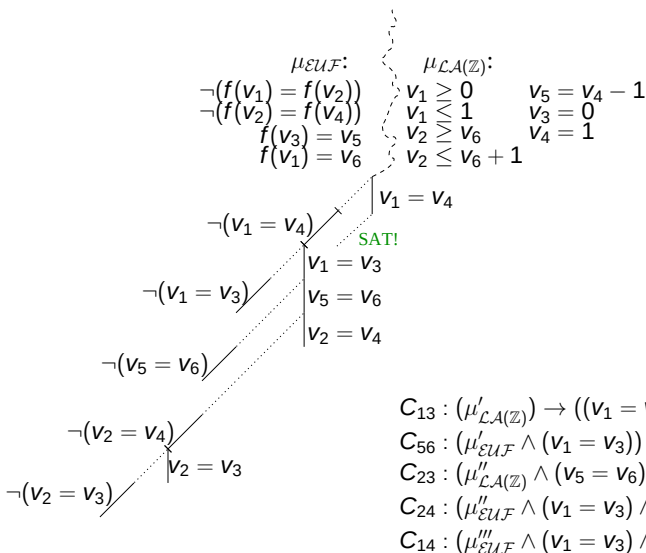
$\mathcal{EUF}$: $(v_3 = h(v_0)) \wedge (v_4 = h(v_1)) \wedge (v_6 = f(v_2)) \wedge (v_7 = f(v_5)) \wedge$

$\mathcal{LA}(\mathbb{Q})$: $(v_0 \geq v_1) \wedge (v_0 \leq v_1) \wedge (v_2 = v_3 - v_4) \wedge (RESET_5 \rightarrow (v_5 = 0)) \wedge$

Both : $(\neg RESET_5 \rightarrow (v_5 = v_8)) \wedge \neg(v_6 = v_7).$



DTC: example (non-convex theory)



Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities**
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Beyond Solving: advanced SAT & SMT functionalities

Advanced SMT functionalities (very important in FV):

- Building *proofs of $\mathcal{T}$ -unsatisfiability*
- Extracting *$\mathcal{T}$ -unsatisfiable Cores*
- Computing *Craig interpolants*
- Performing *All-SMT and Predicate Abstraction*
- Deciding/optimizing *SMT problems with costs*

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 **Beyond Solving: advanced SMT functionalities**
 - **Proofs and unsatisfiable cores**
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Building (Resolution) Proofs of $\mathcal{T}$ -Unsatisfiability

Resolution proof of $\mathcal{T}$ -unsatisfiability

Very similar to building proofs with plain SAT:

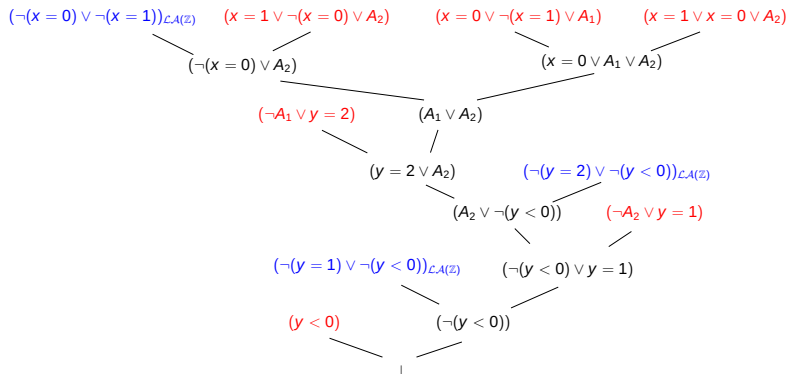
- resolution proofs whose leaves are original clauses and $\mathcal{T}$ -lemmas returned by the $\mathcal{T}$ -solver (i.e., $\mathcal{T}$ -conflict and $\mathcal{T}$ -deduction clauses)
- built by backward traversal of implication graphs, as in CDCL SAT
- Sub-proofs of $\mathcal{T}$ -lemmas can be built in some $\mathcal{T}$ -specific deduction framework if requested

Important for:

- certifying $\mathcal{T}$ -unsatisfiability results
- computing unsatisfiable cores
- computing interpolants

Building Proofs of $\mathcal{T}$ -Unsatisfiability: example

$$(x = 0 \vee \neg(x = 1) \vee A_1) \wedge (x = 0 \vee x = 1 \vee A_2) \wedge (\neg(x = 0) \vee x = 1 \vee A_2) \wedge \\ (\neg A_2 \vee y = 1) \wedge (\neg A_1 \vee x + y > 3) \wedge (y < 0) \wedge (A_2 \vee x - y = 4) \wedge (y = 2 \vee \neg A_1) \wedge (x \geq 0),$$



relevant original clauses, irrelevant original clauses, $\mathcal{T}$ -lemmas

Extraction of $\mathcal{T}$ -unsatisfiable cores

The problem

Given a $\mathcal{T}$ -unsatisfiable set of clauses, extract from it a (possibly small/minimal/minimum) $\mathcal{T}$ -unsatisfiable subset ($\mathcal{T}$ -unsatisfiable core)

- wide literature in SAT
- Some implementations, very few literature for SMT [25, 53]
- We recognize three approaches:
 - *Proof-based* approach (CVCLite, MathSAT):
byproduct of finding a resolution proof
 - *Assumption-based* approach (Yices):
use extra variables labeling clauses, as in the plain Boolean case
 - *Lemma-Lifting* approach [25] :
use an external (possibly-optimized) Boolean unsat-core extractor

The proof-based approach to $\mathcal{T}$ -unsat cores

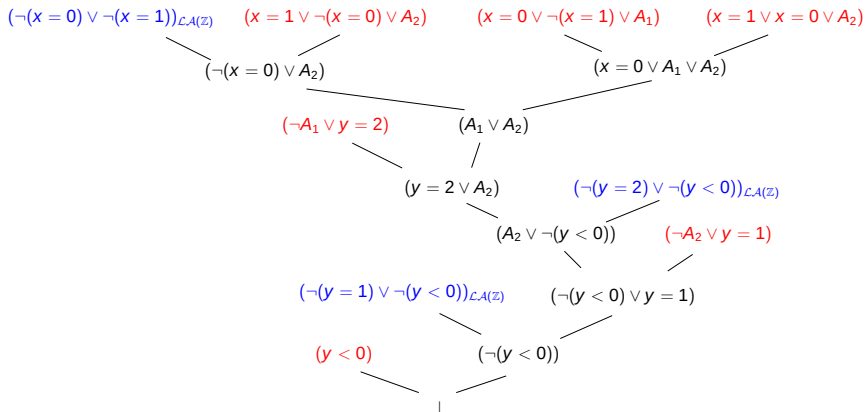
Idea (adapted from [78])

Unsatisfiable core of φ :

- in SAT: the set of leaf clauses of a resolution proof of unsatisfiability of φ
- in $\text{SMT}(\mathcal{T})$: the set of leaf clauses of a resolution proof of $\mathcal{T}$ -unsatisfiability of φ , minus the $\mathcal{T}$ -lemmas

The proof-based approach to $\mathcal{T}$ -unsat cores: example

$$(x = 0 \vee \neg(x = 1) \vee A_1) \wedge (x = 0 \vee x = 1 \vee A_2) \wedge (\neg(x = 0) \vee x = 1 \vee A_2) \wedge \\ (\neg A_2 \vee y = 1) \wedge (\neg A_1 \vee x + y > 3) \wedge (y < 0) \wedge (A_2 \vee x - y = 4) \wedge (y = 2 \vee \neg A_1) \wedge (x \geq 0),$$



The assumption-based approach to $\mathcal{T}$ -unsat cores

Let φ be $\bigwedge_{i=1}^n C_i$ s.t. φ inconsistent.

Idea (adapted from [55])

- 1 each clause C_i in φ is substituted by $\neg S_i \vee C_i$, s.t. S_i fresh “selector” variable
- 2 the resulting formula is checked for **satisfiability under the assumption of all S_i 's**
- 3 final conflict clause at dec. level 0: $\bigvee_j \neg S_j$
 $\implies \{C_j\}_j$ is the unsat core

- extends straightforwardly to $\text{SMT}(\mathcal{T})$.

The assumption-based approach to $\mathcal{T}$ -unsat cores: Example

$$\begin{aligned}
 & (S_1 \rightarrow (x = 0 \vee \neg(x = 1) \vee A_1)) \wedge (S_2 \rightarrow (x = 0 \vee x = 1 \vee A_2)) \wedge \\
 & (S_3 \rightarrow (\neg(x = 0) \vee x = 1 \vee A_2)) \wedge (S_4 \rightarrow (\neg A_2 \vee y = 1)) \wedge \\
 & (S_5 \rightarrow (\neg A_1 \vee x + y > 3)) \wedge (S_6 \rightarrow y < 0) \wedge \\
 & (S_7 \rightarrow (A_2 \vee x - y = 4)) \wedge (S_8 \rightarrow (y = 2 \vee \neg A_1)) \wedge (S_9 \rightarrow x \geq 0)
 \end{aligned}$$

Conflict analysis (Yices 1.0.6) returns:

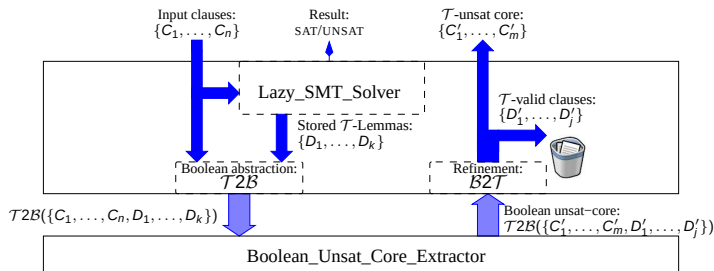
$$\neg S_1 \vee \neg S_2 \vee \neg S_3 \vee \neg S_4 \vee \neg S_6 \vee \neg S_7 \vee \neg S_8,$$

corresponding to the unsat core in red.

The lemma-lifting approach to $\mathcal{T}$ -unsat cores

Idea [25, 29]

- (i) The $\mathcal{T}$ -lemmas D_i are valid in $\mathcal{T}$
- (ii) The conjunction of φ with all the $\mathcal{T}$ -lemmas $D_1, \dots, D_k$ is propositionally unsatisfiable: $\mathcal{T}2\mathcal{B}(\varphi \wedge \bigwedge_{i=1}^n D_i) \models \perp$.



- interfaces with an external Boolean Unsat-core Extractor

$\Rightarrow$ *benefits for free of all state-of-the-art size-reduction techniques*

The lemma-lifting approach to $\mathcal{T}$ -unsat cores: example

$$(x = 0 \vee \neg(x = 1) \vee A_1) \wedge (x = 0 \vee x = 1 \vee A_2) \wedge (\neg(x = 0) \vee x = 1 \vee A_2) \wedge \\ (\neg A_2 \vee y = 1) \wedge (\neg A_1 \vee x + y > 3) \wedge (y < 0) \wedge (A_2 \vee x - y = 4) \wedge (y = 2 \vee \neg A_1) \wedge (x \geq 0),$$

- 1 The SMT solver generates the following set of $\mathcal{LA}(\mathbb{Z})$ -lemmas:

$$\{(\neg(x = 1) \vee \neg(x = 0)), (\neg(y = 2) \vee \neg(y < 0)), (\neg(y = 1) \vee \neg(y < 0))\}.$$

- 2 The following formula is passed to the external Boolean core extractor

$$(B_0 \vee \neg B_1 \vee A_1) \wedge (B_0 \vee B_1 \vee A_2) \wedge (\neg B_0 \vee B_1 \vee A_2) \wedge \\ (\neg A_2 \vee B_2) \wedge (\neg A_1 \vee B_3) \wedge B_4 \wedge (A_2 \vee B_5) \wedge (B_6 \vee \neg A_1) \wedge B_7 \wedge \\ (\neg B_1 \vee \neg B_0) \wedge (\neg B_6 \vee \neg B_4) \wedge (\neg B_2 \vee \neg B_4)$$

which returns the unsat core in red.

- 3 The unsat-core is mapped back, the three $\mathcal{T}$ -lemmas are removed $\implies$ the final $\mathcal{T}$ -unsat core (in red above).

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 **Beyond Solving: advanced SMT functionalities**
 - Proofs and unsatisfiable cores
 - **Interpolants**
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 Conclusions & current research directions

Computing (Craig) Interpolants in SMT

Craig Interpolant

Given an ordered pair (A, B) of formulas such that $A \wedge B \models_{\mathcal{T}} \perp$, a *Craig interpolant* is a formula I s.t.:

- a) $A \models_{\mathcal{T}} I$,
- b) $I \wedge B \models_{\mathcal{T}} \perp$,
- c) $I \preceq A$ and $I \preceq B$.

“ $I \preceq A$ ” meaning that all uninterpreted (in $\mathcal{T}$) symbols in I occur in A .

- Very important in many FV applications
- A few works presented for various theories:
 - $\mathcal{EUF}$ [57, 69], $\mathcal{DL}$ [26, 28], $\mathcal{UTVPI}$ [27, 28], $\mathcal{LA}(\mathbb{Q})$ [57, 69, 26, 28], $\mathcal{LA}(\mathbb{Z})$ [50, 15, 47], $\mathcal{BV}$ [51], ...

A General Algorithm

Algorithm: Interpolant generation for $\text{SMT}(\mathcal{T})$ [67, 57]

- (i) Generate a resolution proof of $\mathcal{T}$ -unsatisfiability $\mathcal{P}$ for $A \wedge B$.
- (ii) ...
- (iii) For every original leaf clause C in $\mathcal{P}$, set $I_C \stackrel{\text{def}}{=} C \downarrow B$ if $C \in A$, and $I_C \stackrel{\text{def}}{=} \top$ if $C \in B$.
- (iv) For every inner node C of $\mathcal{P}$ obtained by resolution from $C_1 \stackrel{\text{def}}{=} p \vee \phi_1$ and $C_2 \stackrel{\text{def}}{=} \neg p \vee \phi_2$, set $I_C \stackrel{\text{def}}{=} I_{C_1} \vee I_{C_2}$ if p does not occur in B , and $I_C \stackrel{\text{def}}{=} I_{C_1} \wedge I_{C_2}$ otherwise.
- (v) Output $I_\perp$ as an interpolant for (A, B) .

“ $\eta \setminus B$ ” [resp. “ $\eta \downarrow B$ ”] is the set of literals in η whose atoms do not [resp. do] occur in B .

• row 2. only place where $\mathcal{T}$ comes in to play

⇒ Reduced to the problem of finding an interpolant for two **sets of $\mathcal{T}$ -literals** (Boolean and $\mathcal{T}$ -specific component decoupled)

A General Algorithm

Algorithm: Interpolant generation for $\text{SMT}(\mathcal{T})$ [67, 57]

- (i) Generate a resolution proof of $\mathcal{T}$ -unsatisfiability $\mathcal{P}$ for $A \wedge B$.
- (ii) **Foreach $\mathcal{T}$ -lemma $\neg\eta$ in $\mathcal{P}$, generate an interpolant $I_{\neg\eta}$ for $(\eta \setminus B, \eta \downarrow B)$.**
- (iii) For every original leaf clause C in $\mathcal{P}$, set $I_C \stackrel{\text{def}}{=} C \downarrow B$ if $C \in A$, and $I_C \stackrel{\text{def}}{=} \top$ if $C \in B$.
- (iv) For every inner node C of $\mathcal{P}$ obtained by resolution from $C_1 \stackrel{\text{def}}{=} p \vee \phi_1$ and $C_2 \stackrel{\text{def}}{=} \neg p \vee \phi_2$, set $I_C \stackrel{\text{def}}{=} I_{C_1} \vee I_{C_2}$ if p does not occur in B , and $I_C \stackrel{\text{def}}{=} I_{C_1} \wedge I_{C_2}$ otherwise.
- (v) Output $I_{\perp}$ as an interpolant for (A, B) .

“ $\eta \setminus B$ ” [resp. “ $\eta \downarrow B$ ”] is the set of literals in η whose atoms do not [resp. do] occur in B .

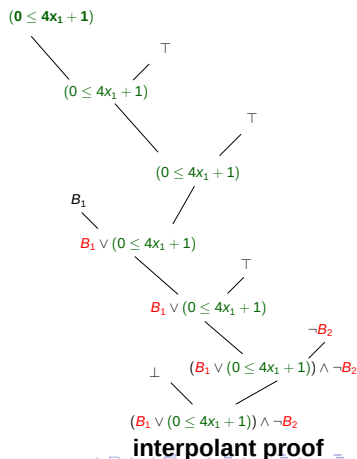
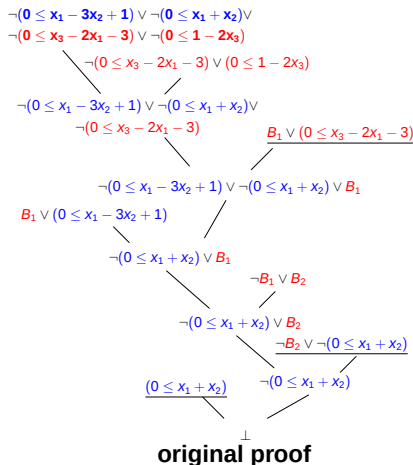
- row 2. only place where $\mathcal{T}$ comes in to play

$\Rightarrow$ Reduced to the problem of finding an interpolant for two **sets of $\mathcal{T}$ -literals** (Boolean and $\mathcal{T}$ -specific component decoupled)

Computing Craig Interpolants in SMT: example

$$A \stackrel{\text{def}}{=} (B_1 \vee (0 \leq x_1 - 3x_2 + 1)) \wedge (0 \leq x_1 + x_2) \wedge (\neg B_2 \vee \neg(0 \leq x_1 + x_2))$$

$$B \stackrel{\text{def}}{=} (\neg(0 \leq x_3 - 2x_1 - 3) \vee (0 \leq 1 - 2x_3)) \wedge (\neg B_1 \vee B_2) \wedge (B_1 \vee (0 \leq x_3 - 2x_1 - 3))$$



Example: interpolation algorithms for difference logic

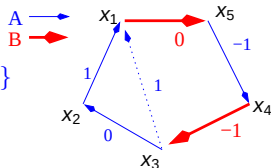
- An inference-based algorithm [57]

$$\begin{array}{c}
 (0 \leq x_1 - x_2 + 1) \quad (0 \leq x_2 - x_3) \\
 \hline
 \text{COMB} \quad (0 \leq x_1 - x_3 + 1) \quad (0 \leq x_4 - x_5 - 1) \\
 \hline
 \text{COMB} \quad (0 \leq x_1 - x_3 + x_4 - x_5) \quad (0 \leq 0) \\
 \hline
 \text{COMB} \quad (0 \leq x_1 - x_3 + x_4 - x_5) \quad (0 \leq 0) \\
 \hline
 \text{COMB} \quad (0 \leq x_1 - x_3 + x_4 - x_5)
 \end{array}$$

$\Rightarrow$ Interpolant: $(0 \leq x_1 - x_3 + x_4 - x_5)$ (not in $\mathcal{DL}$, and weaker).

- A graph-based algorithm [26, 28]

$$\begin{aligned}
 A &\stackrel{\text{def}}{=} \overbrace{\{(0 \leq x_1 - x_2 + 1), (0 \leq x_2 - x_3), (0 \leq x_4 - x_5 - 1)\}}^{\text{Chord: } (0 \leq x_1 - x_3 + 1)} \\
 B &\stackrel{\text{def}}{=} \{(0 \leq x_5 - x_1), (0 \leq x_3 - x_4 - 1)\}.
 \end{aligned}$$



$\Rightarrow$ Interpolant: $(0 \leq x_1 - x_3 + 1) \wedge (0 \leq x_4 - x_5 - 1)$ (still in $\mathcal{DL}$)

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 **Beyond Solving: advanced SMT functionalities**
 - Proofs and unsatisfiable cores
 - Interpolants
 - **All-SMT & Predicate Abstraction**
 - SMT with cost optimization
- 5 Conclusions & current research directions

All-SAT/All-SMT

- *All-SAT*: enumerate all truth assignments satisfying φ
- *All-SMT*: enumerate all $\mathcal{T}$ -satisfiable truth assignments propositionally satisfying φ
- *All-SMT over an “important” subset of atoms* $\mathbf{P} \stackrel{\text{def}}{=} \{P_i\}_i$:
 enumerate all assignments over $\mathbf{P}$ which can be extended to $\mathcal{T}$ -satisfiable truth assignments propositionally satisfying φ
 $\implies$ *can compute predicate abstraction*
- Algorithms:
 - *BCLT* [52]
 each time a $\mathcal{T}$ -satisfiable assignment $\{l_1, \dots, l_n\}$ is found, perform conflict-driven backjumping as if the restricted clause $(\bigvee_i \neg l_i) \downarrow \mathbf{P}$ belonged to the clause set
 - *MathSAT/NuSMV* [23]
 As above, plus the Boolean search of the SMT solver is driven by an OBDD.

Predicate Abstraction

Predicate abstraction

if $\varphi(\mathbf{v})$ is a SMT formula over the domain variables $\mathbf{v} \stackrel{\text{def}}{=} \{v_j\}_j$, $\{\gamma_i\}_i$ is a set of “relevant” predicates over $\mathbf{v}$, and $\mathbf{P} \stackrel{\text{def}}{=} \{P_i\}_i$ a set of Boolean labels, then:

$$\begin{aligned}
 & \text{PredAbs}_{\mathbf{P}}(\varphi) \\
 & \stackrel{\text{def}}{=} \exists \mathbf{v}. (\varphi(\mathbf{v}) \wedge \bigwedge_i P_i \leftrightarrow \gamma_i(\mathbf{v})) \\
 & = \bigvee \left\{ \mu \mid \begin{array}{l} \mu \text{ truth assignment on } \mathbf{P} \\ \text{s.t. } \mu \wedge \varphi \wedge \bigwedge_i (P_i \leftrightarrow \gamma_i) \text{ is } \mathcal{T}\text{-satisfiable} \end{array} \right\}
 \end{aligned}$$

- projection of φ over (the Boolean abstraction of) the set $\{\gamma_i\}_i$.
- essential step in FV: extracts finite-state abstractions from a infinite state space

Predicate Abstraction: example

$$\varphi \stackrel{\text{def}}{=} (v_1 + v_2 > 12)$$

$$\gamma_1 \stackrel{\text{def}}{=} (v_1 + v_2 = 2)$$

$$\gamma_2 \stackrel{\text{def}}{=} (v_1 - v_2 < 10)$$

$$\Downarrow$$

$$\begin{aligned} \text{PreAbs}(\varphi)_{\{P_1, P_2\}} &\stackrel{\text{def}}{=} \exists v_1 v_2 . \left(\begin{array}{l} (v_1 + v_2 > 12) \\ (P_1 \leftrightarrow (v_1 + v_2 = 2)) \\ (P_2 \leftrightarrow (v_1 - v_2 < 10)) \end{array} \wedge \right) \\ &= (\neg P_1 \wedge \neg P_2) \vee (\neg P_1 \wedge P_2) \\ &= \neg P_1. \end{aligned}$$

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 **Beyond Solving: advanced SMT functionalities**
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - **SMT with cost optimization**
- 5 Conclusions & current research directions

SMT cost-minimization problems

The problem

SMT($\mathcal{T}$) problem φ for some $\mathcal{T}$, augmented with cost functions:

$$\text{cost}^i = \sum_{j=1}^{N^i} \text{ite}(P^{ij}, c_1^{ij}, c_2^{ij}), \text{ s.t. } \text{cost}^i \in (l^i, u^i], c_{\{1,2\}}^{ij} > 0$$

- *Decision problem*: is there a model complying with cost ranges?
- *Optimization problem*: find model minimizing some cost^i .
- allows for encoding MaxSAT/MaxSMT and PseudoBoolean

Proposed solution: [63, 24]

- SMT($\mathcal{T} \cup \mathcal{C}$), $\mathcal{C}$ is an ad-hoc “theory of costs”
- a specialized very-fast theory-solver for $\mathcal{C}$ added to MathSAT
 - very fast & aggressive search pruning and theory-propagation
- cost minimization handled by linear or binary search

SMT cost-minimization problems

The problem

SMT($\mathcal{T}$) problem φ for some $\mathcal{T}$, augmented with cost functions:

$$\text{cost}^i = \sum_{j=1}^{N^i} \text{ite}(P^{ij}, c_1^{ij}, c_2^{ij}), \text{ s.t. } \text{cost}^i \in (l^i, u^i], c_{\{1,2\}}^{ij} > 0$$

- *Decision problem*: is there a model complying with cost ranges?
- *Optimization problem*: find model minimizing some cost^i .

- allows for encoding MaxSAT/MaxSMT and PseudoBoolean

Proposed solution: [63, 24]

- **SMT($\mathcal{T} \cup \mathcal{C}$)**, $\mathcal{C}$ is an ad-hoc “theory of costs”
- a specialized very-fast theory-solver for $\mathcal{C}$ added to MathSAT
 - very fast & aggressive search pruning and theory-propagation
- cost minimization handled by linear or binary search

SMT($\mathcal{T} \cup \mathcal{C}$): main ideas

- A “theory of costs” $\mathcal{C}$:
 - Cost variables *cost*^{*i*}
 - “bound cost” *BC*(*cost*^{*i*}, *k*): “*cost*^{*i*} ≤ *k*”
 - “incur cost” *IC*(*cost*^{*i*}, *j*, *k*_{*j*}^{*i*}): “the *j*th addend of *cost*^{*i*} := *k*_{*j*}^{*i*}”
 - “*cost*^{*i*} = $\sum_{j=1}^{N^i} \text{ite}(P_j^i, k_j^i, 0)$, s.t. *cost*^{*i*} ∈ [*l*^{*i*}, *u*^{*i*}]” encoded as
 $\neg BC(\text{cost}^i, l^i) \wedge BC(\text{cost}^i, u^i) \wedge \bigwedge_{j=1}^{N^i} (P_j^i \leftrightarrow IC(\text{cost}^i, j, k_j^i))$
- very-fast theory solver: *C-solver*
 1. *IC*(*cost*^{*i*}, *j*, *k*_{*j*}^{*i*}) = ⊤ $\implies \text{cost}^i = \text{cost}^i + k_j^i$
 2. *cost*^{*i*} > *ub*^{*i*} $\implies$ conflict
 3. *cost*^{*i*} + {total cost of all unassigned *IC*’s} ≤ *lb*^{*i*} $\implies$ conflict
 4. *IC*(*cost*^{*i*}, *j*, *k*_{*j*}^{*i*}) = ⊤ causes 2. $\implies \mathcal{C}$ -propagate $\neg IC(\text{cost}^i, j, k_j^i)$
 5. *IC*(*cost*^{*i*}, *j*, *k*_{*j*}^{*i*}) = ⊥ causes 3. $\implies \mathcal{C}$ -propagate *IC*(*cost*^{*i*}, *j*, *k*_{*j*}^{*i*})
- no symbol shared with $\mathcal{T}$
 $\implies$ independent theory solvers for $\mathcal{T}$ and $\mathcal{C}$

Ongoing work: SMT with linear cost functions

The problem

SMT($\mathcal{LA}(\mathbb{Q}) \cup \bigcup_k \mathcal{T}_k$) problem φ for some $\mathcal{T}_k$'s, augmented with cost functions: $\text{cost}^i = \sum_{j=1}^{N^i} a_j x_j$, s.t. $\text{cost}^i \in (l^i, u^i]$, $x_j \in \mathbb{Q}$

- *Decision problem*: standard SMT($\mathcal{LA}(\mathbb{Q}) \cup \bigcup_k \mathcal{T}_k$)
- *Optimization problem*: find model minimizing some cost^i .
- Optimization along two orthogonal directions: $\mathcal{BOOL}$ and $\mathbb{Q}$
- Allows for encoding **generalized linear disjunctive programming**

Proposed solution:

- SMT($\mathcal{LA}(\mathbb{Q}) \cup \bigcup_k \mathcal{T}_k$) augmented with a LP optimization routine
 - very fast & aggressive search pruning and theory-propagation
- cost minimization (branch&bound or binary search) **embedded inside the CDCL search**

Outline

- 1 Motivations and goals
- 2 Background: Modern SAT solving
 - Basics
 - Modern CDCL SAT solving
 - Other SAT functionalities
- 3 Efficient SMT solving
 - Combining SAT with Theory Solvers
 - Theory Solvers for theories of interest
 - SMT for combinations of theories
- 4 Beyond Solving: advanced SMT functionalities
 - Proofs and unsatisfiable cores
 - Interpolants
 - All-SMT & Predicate Abstraction
 - SMT with cost optimization
- 5 **Conclusions & current research directions**

Conclusions

- SMT very popular, due to successful application in many domains
- Combines techniques from SAT, ATP and operational research
- Not only satisfiability, but also advanced functionalities

Open/ongoing research directions

• Solving:

- **improve efficiency** (e.g. BV , $\mathcal{AR}$, $\mathcal{LA}(\mathbb{Z})$ & their combinations)
*“a never-ending fight against the search-space explosion problem
 [E. Clarke, Turing-award winner 2007]”*
- develop efficient solvers for other theories ($\mathcal{NLA}(\mathbb{Q})$, $\mathcal{NLA}(\mathbb{Z})$)
- develop new theories & solvers (e.g., floating-point arithmetic)
- ...

• Functionalities

- *Interpolation* in some theories ($\mathcal{LA}(\mathbb{Z})$, BV) still very challenging
- *Predicate abstraction (AllSMT)* still a bottleneck in SMT-based FV
- *SMT with costs/optimization* still in very early stage
- ...

- **Combination of SMT solvers and ATP** (SMT with quantifiers)
- Integration & customization of SMT solvers with (FV) tools
- See also [64]

Links I

- a 20-hour PhD course on SAT & SMT:
http://disi.unitn.it/~rseba/DIDATTICA/SAT_BASED10/
- survey papers:
<http://disi.unitn.it/~rseba/publist.html>
 - Roberto Sebastiani: "Lazy Satisfiability Modulo Theories".
 Journal on Satisfiability, Boolean Modeling and Computation, JSAT.
 Vol. 3, 2007. Pag 141–224, ©IOS Press.
 - Clark Barrett, Roberto Sebastiani, Sanjit Seshia, Cesare Tinelli
 "Satisfiability Modulo Theories". Part II, Chapter 26, The Handbook
 of Satisfiability. 2009. ©IOS press.
- web links:
 - The SMT library SMT-LIB:
<http://goedel.cs.uiowa.edu/smtlib/>
 - The SMT Competition SMT-COMP: <http://www.smtcomp.org/>
 - The 2011 MIT SAT/SMT School
<http://people.csail.mit.edu/vganesh/summerschool/>

References I

- [1] A. Armando.
Simplifying OBDDs in Decidable Theories.
In Proc. PDPAR'03., 2003.
- [2] A. Armando, C. Castellini, and E. Giunchiglia.
SAT-based procedures for temporal reasoning.
In Proc. European Conference on Planning, CP-99, 1999.
- [3] G. Audemard, P. Bertoli, A. Cimatti, A. Kornikowicz, and R. Sebastiani.
A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions.
In Proc. CADE'2002., volume 2392 of *LNAI*. Springer, July 2002.
- [4] G. Audemard, P. Bertoli, A. Cimatti, A. Kornikowicz, and R. Sebastiani.
Integrating Boolean and Mathematical Solving: Foundations, Basic Algorithms and Requirements.
In Proc. AIARSC'2002, volume 2385 of *LNAI*. Springer, 2002.
- [5] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani.
Verifying Industrial Hybrid Systems with MathSAT.
In Proc. PDPAR'03, 2003.
- [6] G. Audemard, A. Cimatti, A. Kornikowicz, and R. Sebastiani.
SAT-Based Bounded Model Checking for Timed Systems.
In Proc. FORTE'02., volume 2529 of *LNCS*. Springer, November 2002.
- [7] C. Barret, D. Dill, and A. Stump.
A Generalization of Shostak's Method for Combining Decision Procedures.
In Proc. FROCOS'02, 2002.
- [8] C. Barrett, D. Dill, and A. Stump.
Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT.
In 14th International Conference on Computer-Aided Verification, 2002.

References II

- [9] C. Barrett, R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Splitting on Demand in SAT Modulo Theories.
In *Proc. LPAR'06*, volume 4246 of *LNAI*. Springer, 2006.
- [10] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli.
Satisfiability Modulo Theories.
In *Handbook of Satisfiability*, chapter 26, pages 825–885. IOS Press, 2009.
- [11] P. Baumgartner.
FDPLL - A First Order Davis-Putnam-Longeman-Loveland Procedure.
In *Proceedings of CADE-17*, pages 200–219. Springer-Verlag, 2000.
- [12] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, P. van Rossum, S. Ranise, and R. Sebastiani.
Efficient Satisfiability Modulo Theories via Delayed Theory Combination.
In *Proc. CAV 2005*, volume 3576 of *LNCS*. Springer, 2005.
- [13] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, S. Ranise, P. van Rossum, and R. Sebastiani.
Efficient theory combination via boolean search.
Information and Computation, 204(10):1493–1525, 2006.
- [14] M. Bozzano, R. Bruttomesso, A. Cimatti, T. A. Junttila, P. van Rossum, S. Schulz, and R. Sebastiani.
Mathsat: Tight integration of sat and mathematical decision procedures.
Journal of Automated Reasoning, 35(1-3):265–293, 2005.
- [15] A. Brillout, D. Kroening, P. Rümmer, and T. Wahl.
An Interpolating Sequent Calculus for Quantifier-Free Presburger Arithmetic.
In *Proc. IJCAR*, volume 6173 of *LNCS*. Springer, 2010.
- [16] R. Brinkmann and R. Drechsler.
RTL-datapath verification using integer linear programming.
In *Proc. ASP-DAC 2002*, pages 741–746. IEEE, 2002.

References III

- [17] R. Brummaryer and A. Biere.
Lemmas on Demand for the Extensional Theory of Arrays.
Journal on Satisfiability, Boolean Modeling and Computation (JSAT), 6, 2009.
- [18] R. Brummayer and A. Biere.
Boolelector: An efficient smt solver for bit-vectors and arrays.
In *TACAS*, volume 5505 of *LNCS*, pages 174–177. Springer, 2009.
- [19] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, Z. Hanna, A. Nadel, A. Palti, and R. Sebastiani.
A Lazy and Layered SMT($\mathcal{BV}$) Solver for Hard Industrial Verification Problems.
In *CAV*, volume 4590 of *LNCS*, pages 547–560. Springer, 2007.
- [20] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani.
Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: A Comparative Analysis.
In *Proc. LPAR*, volume 4246 of *LNCS*. Springer, 2006.
- [21] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, and R. Sebastiani.
Delayed Theory Combination vs. Nelson-Oppen for Satisfiability Modulo Theories: A Comparative Analysis. .
Annals of Mathematics and Artificial Intelligence., 55(1-2), 2009.
- [22] J. R. Burch and D. L. Dill.
Automatic Verification of Pipelined Microprocessor Control.
In *Proc. CAV '94*, volume 818 of *LNCS*. Springer, 1994.
- [23] R. Cavada, A. Cimatti, A. Franzén, K. Kalyanasundaram, M. Roveri, and R. K. Shyamasundar.
Computing Predicate Abstractions by Integrating BDDs and SMT Solvers.
In *FMCAD*, pages 69–76. IEEE Computer Society, 2007.
- [24] A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, and C. Stenico.
Satisfiability modulo the theory of costs: Foundations and applications.
In *TACAS*, volume 6015 of *LNCS*, pages 99–113. Springer, 2010.

References IV

- [25] A. Cimatti, A. Griggio, and R. Sebastiani.
A Simple and Flexible Way of Computing Small Unsatisfiable Cores in SAT Modulo Theories.
In *SAT*, volume 4501 of *LNCS*, pages 334–339. Springer, 2007.
- [26] A. Cimatti, A. Griggio, and R. Sebastiani.
Efficient Interpolant Generation in Satisfiability Modulo Theories.
In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08.*, volume 4963 of *LNCS*. Springer, 2008.
- [27] A. Cimatti, A. Griggio, and R. Sebastiani.
Interpolant Generation for UTVPI.
In *CADE*, volume 5663 of *LNCS*, pages 167–182, 2009.
- [28] A. Cimatti, A. Griggio, and R. Sebastiani.
Efficient Generation of Craig Interpolants in Satisfiability Modulo Theories.
ACM Transaction on Computational Logics – TOCL, 12(1), October 2010.
- [29] A. Cimatti, A. Griggio, and R. Sebastiani.
Computing small unsatisfiable cores in sat modulo theories.
Journal of Artificial Intelligence Research, JAIR, 40:701–728, April 2011.
- [30] S. Cotton and O. Maler.
Fast and Flexible Difference Logic Propagation for DPLL(T).
In *Proc. SAT'06*, volume 4121 of *LNCS*. Springer, 2006.
- [31] M. Davis, G. Longemann, and D. Loveland.
A machine program for theorem proving.
Journal of the ACM, 5(7), 1962.
- [32] M. Davis and H. Putnam.
A computing procedure for quantification theory.
Journal of the ACM, 7:201–215, 1960.

References V

- [33] T. B. de la Tour.
Minimizing the Number of Clauses by Renaming.
In Proc. of the 10th Conference on Automated Deduction, pages 558–572. Springer-Verlag, 1990.
- [34] L. de Moura and N. Bjørner.
Efficient E-matching for SMT solvers.
In Proc. CADE-21, 21st International Conference on Automated Deduction, volume 4603 of *LNCS*. Springer, 2007.
- [35] L. de Moura, H. Ruess, and M. Sorea.
Lazy Theorem Proving for Bounded Model Checking over Infinite Domains.
In Proc. CADE'2002., volume 2392 of *LNAI*. Springer, July 2002.
- [36] L. M. de Moura and N. Bjørner.
Model-based theory combination.
Electr. Notes Theor. Comput. Sci., 198(2):37–49, 2008.
- [37] L. M. de Moura and N. Bjørner.
Generalized, efficient array decision procedures.
In FMCAD, pages 45–52. IEEE, 2009.
- [38] D. Detslefs, G. Nelson, and J. Saxe.
Simplify: a theorem prover for program checking.
Journal of the ACM, 52(3):365–473, 2005.
- [39] B. Dutertre and L. de Moura.
System Description: Yices 1.0.
In Proc. on 2nd SMT competition, SMT-COMP'06, 2006.
Available at yices.csl.sri.com/yices-smtcomp06.pdf.
- [40] N. Eén and N. Sörensson.
An extensible SAT-solver.
In Theory and Applications of Satisfiability Testing (SAT 2003), volume 2919 of *LNCS*, pages 502–518. Springer, 2004.

References VI

- [41] A. Franzen, A. Cimatti, A. Nadel, R. Sebastiani, and J. Shalev.
Applying SMT in Symbolic Execution of Microcode.
In Proc. Int. Conference on Formal Methods in Computer Aided Design (FMCAD'10). IEEE, 2010.
- [42] V. Ganesh and D. L. Dill.
A Decision Procedure for Bit-Vectors and Arrays.
In CAV, 2007.
- [43] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
DPLL(T): Fast decision procedures.
In R. Alur and D. Peled, editors, Proceedings of the 16th International Conference on Computer Aided Verification, CAV'04 (Boston, Massachusetts), LNCS. Springer, 2004.
(To appear).
- [44] F. Giunchiglia and R. Sebastiani.
Building decision procedures for modal logics from propositional decision procedures - the case study of modal K.
In Proc. CADE'13, LNAI, New Brunswick, NJ, USA, August 1996. Springer.
- [45] A. Goel, S. Krstić, and A. Fuchs.
Deciding array formulas with frugal axiom instantiation.
In Proceedings of SMT'08/BPR'08, pages 12–17, New York, NY, USA, 2008. ACM.
- [46] A. Griggio.
A Practical Approach to SMT(LA(Z)).
In Proc. SMT 2010, 2010.
- [47] A. Griggio, T. T. H. Le, and R. Sebastiani.
Efficient Interpolant Generation in Satisfiability Modulo Linear Integer Arithmetic.
In Proc. Tools and Algorithms for the Construction and Analysis of Systems, TACAS'11, LNCS. Springer, 2011.

References VII

- [48] A. Griggio, Q. S. Phan, R. Sebastiani, and S. Tomasi.
Stochastic Local Search for SMT: Combining Theory Solvers with WalkSAT.
In *Frontiers of Combining Systems, FroCoS'11*, LNAI. Springer, 2011.
- [49] I. Horrocks and P. F. Patel-Schneider.
FaCT and DLP.
In *Proc. Tableaux'98*, pages 27–30, 1998.
- [50] H. Jain, E. M. Clarke, and O. Grumberg.
Efficient Craig Interpolation for Linear Diophantine (Dis)Equations and Linear Modular Equations.
In A. Gupta and S. Malik, editors, *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 254–267. Springer, 2008.
- [51] D. Kroening and G. Weissenbacher.
Lifting Propositional Interpolants to the Word-Level.
In *FMCAD*, pages 85–89, Los Alamitos, CA, USA, 2007. IEEE Computer Society.
- [52] S. K. Lahiri, R. Nieuwenhuis, and A. Oliveras.
SMT techniques for fast predicate abstraction.
In *Proc. CAV, LNCS 4144*. Springer, 2006.
- [53] M. Liffiton and K. Sakallah.
Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints.
Journal of Automated Reasoning, 40(1), 2008.
- [54] I. Lynce and J. Marques-Silva.
On Computing Minimum Unsatisfiable Cores.
In *7th International Conference on Theory and Applications of Satisfiability Testing*, 2004.
- [55] I. Lynce and J. P. Marques-Silva.
On computing minimum unsatisfiable cores.
In *SAT*, 2004.

References VIII

- [56] M. Mahfoudh, P. Niebert, E. Asarin, and O. Maler.
A Satisfiability Checker for Difference Logic.
In Proceedings of SAT-02, pages 222–230, 2002.
- [57] K. L. McMillan.
An interpolating theorem prover.
Theor. Comput. Sci., 345(1):101–121, 2005.
- [58] J. Moeller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard.
Fully symbolic model checking of timed systems using difference decision diagrams.
In Proc. Workshop on Symbolic Model Checking (SMC), FLoC'99, Trento, Italy, July 1999.
- [59] M. W. Moskewicz, C. F. Madigan, Y. Z., L. Zhang, and S. Malik.
Chaff: Engineering an efficient SAT solver.
In Design Automation Conference, 2001.
- [60] G. Nelson and D. Oppen.
Simplification by Cooperating Decision Procedures.
ACM Trans. on Programming Languages and Systems, 1(2):245–257, 1979.
- [61] R. Nieuwenhuis and A. Oliveras.
Congruence closure with integer offsets.
In In 10th Int. Conf. Logic for Programming, Artif. Intell. and Reasoning (LPAR), volume 2850 of *LNAI*, pages 78–90. Springer, 2003.
- [62] R. Nieuwenhuis and A. Oliveras.
DPLL(T) with Exhaustive Theory Propagation and its Application to Difference Logic.
In Proc. CAV'05, volume 3576 of *LNCS*. Springer, 2005.

References IX

- [63] R. Nieuwenhuis and A. Oliveras.
On SAT Modulo Theories and Optimization Problems.
In Proc. Theory and Applications of Satisfiability Testing - SAT 2006, volume 4121 of *LNCS*. Springer, 2006.
- [64] R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio.
Challenges in Satisfiability Modulo Theories.
In Proc. RTA'07, volume 4533 of *LNCS*. Springer, 2007.
- [65] R. Nieuwenhuis, A. Oliveras, and C. Tinelli.
Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T).
Journal of the ACM, 53(6):937–977, November 2006.
- [66] D. Plaisted and S. Greenbaum.
A Structure-preserving Clause Form Translation.
Journal of Symbolic Computation, 2:293–304, 1986.
- [67] P. Pudlák.
Lower bounds for resolution and cutting planes proofs and monotone computations.
J. of Symb. Logic, 62(3), 1997.
- [68] H. Rueß and N. Shankar.
Deconstructing Shostak.
In Proc. LICS '01. IEEE Computer Society, 2001.
- [69] A. Rybalchenko and V. Sofronie-Stokkermans.
Constraint Solving for Interpolation.
In Proc. VMCAI, volume 4349 of *LNCS*. Springer, 2007.
- [70] R. Sebastiani.
Lazy Satisfiability Modulo Theories.
Journal on Satisfiability, Boolean Modeling and Computation, JSAT, 3(3-4):141–224, 2007.

References X

- [71] R. Shostak.
Deciding Combinations of Theories.
Journal of the ACM, 31:1–12, 1984.
- [72] J. P. M. Silva and K. A. Sakallah.
GRASP - A new Search Algorithm for Satisfiability.
In *Proc. ICCAD'96*, 1996.
- [73] K. Stergiou and M. Koubarakis.
Backtracking algorithms for disjunctions of temporal constraints.
In *Proc. AAAI*, pages 248–253, 1998.
- [74] A. Stump, C. W. Barrett, and D. L. Dill.
CVC: A Cooperating Validity Checker.
In *Proc. CAV'02*, number 2404 in LNCS. Springer Verlag, 2002.
- [75] S. Wolfman and D. Weld.
The LPSAT Engine & its Application to Resource Planning.
In *Proc. IJCAI*, 1999.
- [76] T. Yavuz-Kahveci, M. Tuncer, and T. Bultan.
A Library for Composite Symbolic Representation.
In *Proc. TACAS2001*, volume 2031 of LNCS. Springer Verlag, 2000.
- [77] L. Zhang, C. F. Madigan, M. W. Moskewicz, and S. Malik.
Efficient conflict driven learning in boolean satisfiability solver.
In *ICCAD*, pages 279–285, 2001.
- [78] L. Zhang and S. Malik.
Extracting small unsatisfiable cores from unsatisfiable boolean formula.
In *Proc. of SAT*, 2003.

References XI

Disclaimer

The list of references above is by no means intended to be all-inclusive. I apologize both with the authors and with the readers for all the relevant works which are not cited here.



©Warner Bros. Inc.